Zoomable Visualizations of Big Skewed Spatial Data

Wenbo Tao, Leilani Battle, Remco Chang, Michael Stonebraker

Abstract— Zoomable spatial visualizations (ZSV) have shown to be effective in reducing visual clutter and occlusion on big skewed datasets, which has been the Achilles heel of static spatial visualizations. However, programming ZSVs is a daunting task with existing tools/systems. First, many ZSV systems assume that data fits in memory, fetching data in user's viewport on-the-fly. Therefore, they cannot scale to big skewed datasets. Second, majority of the ZSV systems focus on only a subset of the ZSV design space, inevitably leading to low flexibility. Lastly, general pan/zoom toolkits do not assist the developer with managing visual clutter and occlusion, which is a very challenging task especially on big datasets. In this paper, we describe AutoDD, a work-in-progress system for easy construction of ZSVs at scale. AutoDD adopts a declarative model that captures a large design space, allowing easy specifications of ZSVs that are suitable for a variety of ZSV tasks. Behind the scenes, AutoDD efficiently calculates the layout of objects in the multi-scale zooming space that satisfies occlusion and density constraints. To achieve interactive pan/zoom rates, we integrate AutoDD with Kyrix, a recent system for creating large-scale general zoomable visualizations.

1 Introduction

Spatial data visualizations such as attraction maps and scatterplots are an essential component of exploratory visualization systems. By visualizing data items on a two dimensional cartesian plane decorated with visual channels such axes and background maps, these visualizations provide domain analysts with positional contexts of the underlying data, allowing them to both inspect individual visual objects and find global patterns that drive important decision making.

For big and skewed datasets, visualizing every single data item inevitably leads to overcrowded display. To address visual clutter, there has been substantial research [22, 17, 23, 19, 21, 13, 12] on devising aggregated visualizations (e.g. density maps and bar charts). While aggregated visualizations are free of visual clutter, they lack the functionality to query individual objects, which is often crucial in many tasks performed in spatial visualizations [27, 9, 24]. Prior research also studied how to use transparency [10, 15], animation [6] and displacement of visual objects [30, 14, 29] to ease the overdraw problem. However, due to limited screen resolution and human perception ability, these approaches possess limited scalability.

Zoomable spatial visualizations (ZSV) have shown the potential to mitigate visual clutter, while allowing many tasks typically performed on spatial visualizations such as interaction with individual visual objects and focused exploration within a neighborhood [27]. By expanding the two dimensional plane into a multi-scale zooming space, more screen resolution becomes available, allowing visual objects to be placed in a way that possibly avoids occlusion and excessive density. The user of a ZSV can use pan and zoom to conveniently navigate this large multi-scale data space to get either an overview of the underlying data or a focused inspection of an area of interest. For example, Figure 2c is a ZSV of NBA basketball games constructed by the system we describe in this paper. The user can see a few exemplary games on the top zoom level, and zoom in to see more games.

Despite the utility of ZSVs, programming such visualizations remains a daunting task for application developers, especially when the data is large and skewed. The reasons are multifold.

Firstly, tools for creating general zoomable visualizations (e.g. Kyrix [28], Pad++ [2], Jazz [3] and ZVTM [26]) require the developer to manually generate the object layouts in the multi-scale zooming space (e.g. what visual objects appear on what zoom level) to manage clutter and occlusion on his/her own. This is a challenging task especially for big datasets [31, 11, 16]. Therefore, users of these systems inevitably experience prolonged development and likely unsatisfying end results.

Secondly, big skewed datasets pose significant challenges to the scalability of existing ZSV systems. Many systems [31, 5, 20, 16, 11] assume that data fits in memory, retrieving visual objects in the user's viewport on-the-fly. This assumption is not practical when the dataset is highly skewed and millions of visual objects can be in the viewport at the same time. Some systems precompute [5, 16, 8, 4] the object layouts of the ZSV, but the precomputation algorithms are not parallelized and thus cannot scale to large datasets.

Lastly, many prior ZSV systems/tools focus on only a subset of the ZSV design space, preventing the developer from flexibly making application-specific design choices. For example, much attention has been drawn to specific types of visual object encodings, e.g., small-sized dots [8, 31, 5], heatmap [25, 7], glyph-based encodings [16, 4] and contours [20]. Some works focus on object overlap removal [4, 5] while others center on density limits [8, 25, 11, 32]. The narrow focuses make these tools hard to extend to general scenarios, require the developer to constantly switch tools for different application requirements and stymie the creation of customized visualizations.

These limitations of existing systems strongly motivate for an integrated system that supports easy specification of a wide range of design choices of ZSVs, and automate the creation of ZSVs at scale.

In this paper, we describe our ongoing effort in building AutoDD, a system for easy creation and exploration of ZSVs. Our goal is to provide the developer with a concise declarative specification model that is able to express a variety of ZSVs. Behind the scenes, we run automated algorithms to decide the object layouts in the multi-scale zooming space. To enable interactive browsing of the generated ZSVs, we integrate our system with Kyrix [28], a recent system for creating general zoomable visualizations of large datasets.

The rest of the paper is organized as follows. We first give an overview of the system in Section 2. We then describe in Section 3 our declarative specification model and example ZSVs. Section 4 presents a hierarchical sampling and aggregation algorithm that generates the object layouts of ZSVs.

- *Wenbo Tao is with Massachusetts Institute of Technology. E-mail: wenbo@mit.edu.*
- *Leilani Battle is with University of Maryland, College Park. E-mail: leilani@cs.umd.edu.*
- *Remco Chang is with Tufts University. E-mail: remco@cs.tufts.edu.*
- *Michael Stonebraker is with MIT. E-mail: stonebraker@csail.mit.edu.*

2 System Overview

We begin with overviewing the architecture, supported tasks and design goals of our system.

2.1 Architecture

Figure 1 shows the architecture of AutoDD. We build AutoDD on top of Kyrix [28], a recent system we built for authoring general zoomable visualizations at scale. Despite its generality (e.g. supporting general data, more types zooming transitions), Kyrix does not support ZSVs

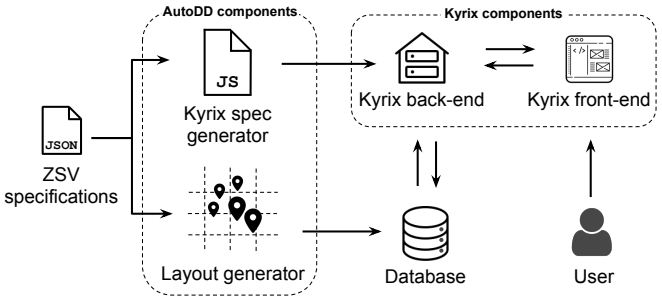


Figure 1. AutoDD system architecture. AutoDD offers declarative specification of ZSVs, and generates Kyrix [28] applications to achieve interactive online performance at scale.

very well. Specifically, Kyrix leaves to the developer the burden of generating object layouts of ZSVs, which is arguably a challenging research topic for more than a decade. Furthermore, Kyrix's declarative grammar requires verbose specification of individual zoom levels, which should be abstracted away by a more concise grammar.

These two limitations of Kyrix have strongly motivated the design of AutoDD. We aim to provide the developer with a concise declarative grammar for ZSVs, automate the layout generation, and use Kyrix's spatial indexing to achieve interactive pan and zoom.

Overall, AutoDD allows the developer to declaratively specify ZSVs in the JSON format, and serves as a Kyrix application generator. We have integrated AutoDD into the open source Kyrix system[1] to provide a set of high-level APIs for ZSV authoring.

Here, we first briefly present necessary background information on Kyrix. Interested readers can refer to the paper [28] for more details. Then we describe components of AutoDD in more details.

Kyrix. Kyrix offers declarative specification of general zoomable visualizations. The developer uses *canvas*es to model different levels of details in a zoomable visualization, and uses *zoom*s to indicate that one can zoom in/out from one canvas to another. Each canvas is consisted of one or more overlaid *layer*s, each of which is associated with the following:

- One *database query* which the Kyrix back-end uses to fetch data items from a database.
- One *rendering function* which the Kyrix front-end runs to convert data items to visual objects.
- One *placement function* which calculates a bounding box for each data item. The Kyrix back-end builds database spatial indexes on these bounding boxes, and fetch data on demand at runtime by querying data items whose bounding boxes intersect with user's viewport.

AutoDD's layout generator. A layout generator computes the placements of visual objects in the multi-scale zooming space and stores them as database tables. Each database table serves as the data source for one layer in the Kyrix application. Details of the layout algorithms are in Section 4.

AutoDD's Kyrix specification generator. Given a ZSV specification, the Kyrix specification generator generates Kyrix specifications that can be turned into interactive web-based visualizations. Generating canvases and zooms is relatively straightforward. For each layer, the database query is simply selecting everything from the corresponding database table produced by the layout generator. The rendering and placement functions are based on the value of various parameters in the ZSV specification, e.g., visual object encodings, whether showing outliers, bandwidth of density functions, etc. Details can be found in Section 3.

2.2 Design Goals

Tasks and expressivity. We identify a set of ZSV tasks that AutoDD should support on the basis of a recent survey of scatterplot tasks [27] which summarized twelve basic tasks that the user typically performs with scatterplots (Table 1). Based on our knowledge of the ZSV literature and experiences with ZSV users, we augment this task set with two more tasks that are specific to ZSVs: browsing exemplary objects and inspect aggregated metrics.

Table 1. ZSV tasks supported in AutoDD, based on the scatterplot task survey [27]. We augmented the original task set with two additional tasks (in bold) for ZSVs.

Object-centric	Identify object
	Locate object
	Verify object
	Object comparison
Browsing	Explore neighborhood
	Search for known motif (cluster, correlation)
	Look for global trends
	Browse exemplary objects
Aggregate-level	Characterize distribution
	Identify anomalies
	Identify correlation
	Density comparison
	Undestand distances
	Inspect aggregated metrics

Our first goal (**G1**) is to enable the developer to easily specify ZSV designs that are appropriate for any desired task in Table 1. Although it is possible to support more than one task in a single ZSV, we do not attempt to enumerate all combinations here since there are too many of them. Instead, we focus on supporting one task at a time, and explore ways to support multiple tasks by overlaying multiple ZSVs.

According to the survey by Sarikaya et al [27], different design choices often have different appropriateness for a given task. Therefore, by accomodating different tasks, we naturally achieve reasonable expressivity in design choices.

Scalability. Our second goal (**G2**) is to ensure the scalability of AutoDD. This goal has three subgoals. The first one is human perception scalability (**G2-a**): users of the ZSVs generated by AutoDD should not be burdend by too much information at any viewing region. Second, although the layout generation is performed offline, it must scale to large datasets (**G2-b**). More precisely, its time complexity must be at most $O(n \log n)$ where n is the number of total data items, and should be parallelized. Third, the end-to-end response times to any user interaction (pan or zoom) must be under 500 ms (**G2-c**), an emprical upper bound that ensures fluid interactions [18].

2.3 Current Status

Currently, we have an intial AutoDD system running, with an incomplete declarative model that covers a subset of the tasks in Table 1, and an $O(n \log n)$ layout generator algorithm that runs on a single node.

The following two sections describe the current system in greater detail and present our plans towards fully fulfilling **G1** and **G2**.

3 Declarative Model and Example ZSVs

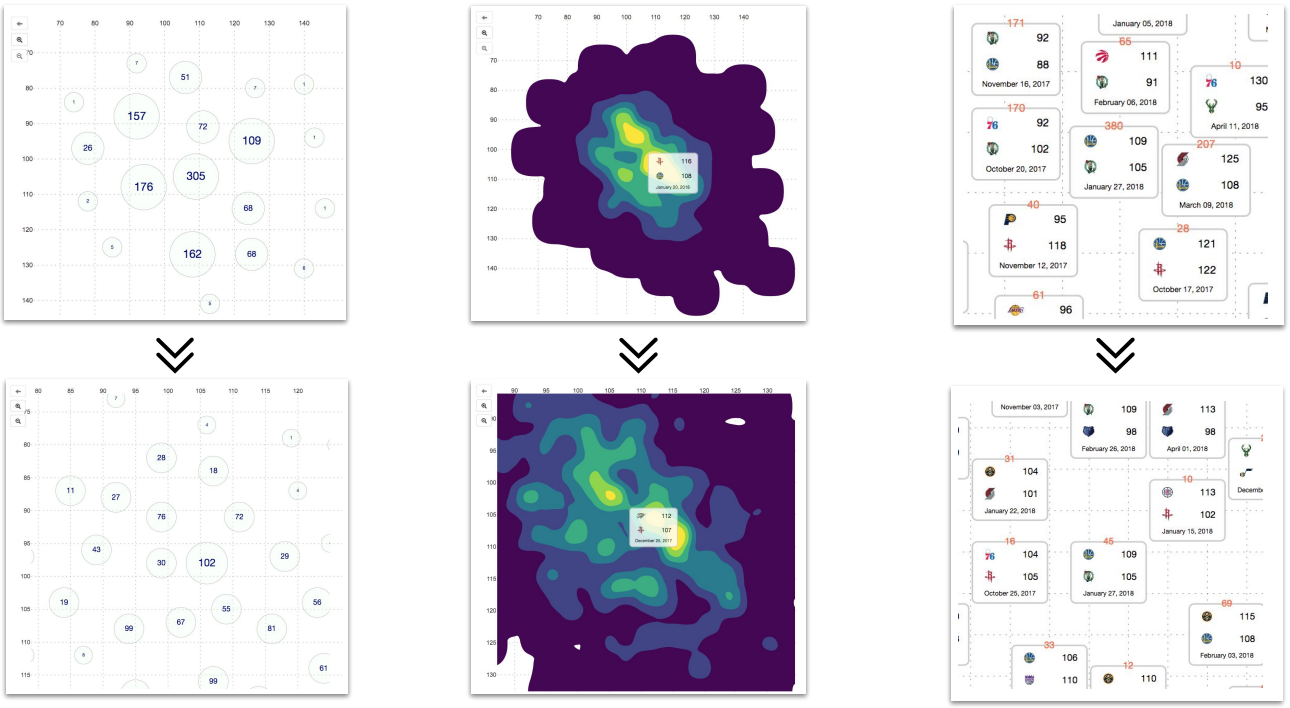
G1 motivates us to design a concise and expressive declarative model that captures a variety of design choices for different tasks.

By reviewing ZSVs in both the literature and commercial systems, we identify a set of high-level design choices that form basic components of AutoDD's declarative model.

Data. We assume that raw data items reside in a database and should be specified as a database query. AutoDD's layout generator fetches raw data items using this query and generates "cooked" data items with additional information such as screen coordinates and aggregated densities.

X & Y placement. This refers to the X and Y components of the placements of objects within the multi-scale zooming space. To automate the calculation of these components, our declarative model only asks the developer for two fields of raw data items (e.g. longitude and latitude) that represent where the objects locate if every object is

[1] https://www.github.com/tracyhenry/kyrix



```
{
    "data": "SELECT * FROM GAMES;",
    "x": "home score",
    "y": "away_score",
    "axis": true,
    "mark": "circle"
}
```

(a)

```
{
    "data": "SELECT * FROM GAMES;",
    "x": "home_score",
    "y": "away score",
    "z": "agg_ranking",
    "axis": true,
    "overlap": true,
    "mark": "contour+object",
    "renderer": "{custom renderer}"
}
```

(b)

```
{
    "data": "SELECT * FROM GAMES;",
    "x": "home score",
    "y": "away_score",
    "z": "agg ranking",
    "axis": true,
    "mark": "object+numerosity",
    "renderer": "{custom renderer}"
}
```

(c)

Figure 2. Three example applications and their AutoDD specifications. They are all visualizations of 2017–2018 season NBA basketball games, with X axis being the score of the home team and Y axis being the score of the away team: (a) circles with aggregated numbers showing the number of objects around that area. The radius and font size of a circle is proportional to the number inside it; (b) contour lines showing the 2D density distribution, with lighter colors indicating higher densities. The user can hover over the contour lines to see example games rendered using a custom renderer. Games between teams with higher aggregated ranking are placed on top zoom levels. Overlap is allowed; (c) games are rendered using a custom renderer, with aggregated numbers indicating numerosity. Games do not overlap.

drawn on a single-view static spatial visualization. The object layout generator then takes care of the scaling of these attributes. Currently we are focused on exact X/Y placements (i.e. X/Y are raw coordinates scaled by a scaling factor). In the future, we plan to consider inexact placements, e.g. binning the 2D canvas space or slight displacements of the objects.

Z placement. The Z placements of objects are controlled by an importance metric, specified as one field from raw data items. The layout generator places more important objects above less important objects in the multi-scale zooming space, i.e., important objects appear in overviews and less important objects are revealed as the user zooms into detailed views.

Visual encodings. This refers to how "cooked" data items are converted to visual objects on the screen. AutoDD provides a library of such rendering templates, including (currently) thumbails, numbered circles (Figure 2a) and contour density polygons (Figure 2b). We also allow custom rendering functions (Figure 2c). We plan to allow specifications of increasingly more detailed visual representations (e.g. employee thumbnails in overviews then detailed profile boxes in detailed views).

Overlap. This controls whether object overlaps are allowed.

Overlay. Overlay allows two types of visual encodings to be visible at the same time, which could enable a ZSV to be suitable for more tasks. For example, overlaying transparent thumbnails over contour density polygons is suitable for both "density comparison" and "browse exemplary objects" (Table 1). Neither encoding alone is suitable for both tasks.

Figure 2 shows several examples constructed using the current AutoDD. These visualizations all feature multiple levels of details arranged in multiple zoom levels. As the user zooms in, either more objects or finer-grained aggregated marks become visible. These are backed by a hierarchical sampling and aggregation algorithm which we describe in the next section.

4 Layout Algorithm

In this section, we present the algorithm that the layout generator employs to place objects in the multi-scale zooming space.

To simplify the problem, we assume that multi-scale zooming space is consisted of discrete zoom levels, where the levels are numbered 0, 1, 2 ... from top to bottom, and the zoom factor between adjacent levels is a constant (e.g. 2 as in many web maps). Having discrete zoom levels has proven to be an universally accessible paradigm in many visualization systems [28, 1]. It also frees us from optimizing in a large and continuous search space.

This layout algorithm is divided into two phases. Phase I (Section 4.1) is a top-down sampling process that decides the placement of objects in the multi-scale zooming space. Phase II (Section 4.2) computes necessary aggregation information, e.g., aggregated densities, averages and convex hulls, in a bottom up fashion. Both phases add additional fields to raw data items, which can be used by corresponding renderers to produce ZSVs.

4.1 Top Down Sampling

Phase I determines what objects appear on what zoom levels. Based on the goals we have, we identify several constraints on the placement of the objects:

- *Zooming continuity.* We enforce that if an object appears on one zoom level, it always appear on zoom levels that are below this level. So Phase I essentially determines for each object on which zoom level it starts to appear. This "zooming continuity" is crucial to making a ZSV usable [31].
- *Overlap constraint.* If non-overlapping is specified, objects should not overlap with each other. We assume there is a constant-sized bounding box associated with each object, either built in with design templates or specified by the developer, and only check the overlap of bounding boxes for simplicity.
- *Density constraint.* We impose an upper limit on the number of objects in any 1K by 1K (in pixels) window on any zoom level. Currently, this limit is determined based on empirical estimates of how many objects Kyrix's front-end and back-end can process at the same time without compromising performance (**G2-c**). We plan to look into human perception studies to get an upper limit based on visual encoding complexity (**G2-a**), compare it with the current limit and then choose the smaller one.

With these constraints, we optimize for two goals. First, we want to place more important objects on top zoom levels if an importance metric is specified. Second, we want to set up as few zoom levels as possible (with the bottom level having all objects) to reduce the amount of interaction required to reach the finest level of details.

We do not propose quantitative optimization metrics based on our constraints and goals due to the potential complexity involved. A prior work [31] has proposed an optimization metric with only a subset of our constraints and goals, and has proven that finding the optimal object placement is NP-hard. Therefore, we keep our optimization goals qualitative and look for heuristic solutions.

A greedy-based sampling procedure. We perform the same sampling procedure for each level, with the samples of level i being the *initial samples* of level $i+1$ (the top level has no initial samples). This ensures zooming continuity.

For each level, we iterate through all objects in their importance order (if not specified, in any order). For each object considered, we check if adding it violates the density constraint or the overlap constraint. If not, we add it to the sample set.

To check density and overlap constraints while trying to add an object, we make use of a parameter θ, the ratio of the minimum X (Y) distance between the centroids of any two data items to the width (height) of the bounding box. The higher θ is, more spacing there is between visual objects, and more likely it is that the density constraint is satisfied. Having "fewest zoom levels in mind", we therefore choose the smallest θ that satisfies the density constraint, which can be derived mathematically. Note that if non-overlapping is specified, θ must be at least 1.

The key challenge is how to make sure that the distance between samples is as what θ dictates in an efficient way. We enlarge the bounding boxes of objects by a factor of θ. We then maintain an R-tree of the bounding boxes of existing samples. Every time we try to sample a data item, we use the R-tree to test if the enlarged bounding box of the new object overlaps with any existing bounding boxes. If not, we add the bounding box of the new data item into the R-tree.

After this top-down sampling procedure, we produce a series of zoom levels of data items, where each zoom level is stored in one database table. Each data item is augmented with the X & Yplacement information (note that Z is encoded by the database table). Also note that a data item is duplicated to every zoom level where the associated visual object is visible.

4.2 Bottom Up Aggregation

The objective of Phase II is to augment the data items produced in Phase I with aggregation information that are necessary for aggregate visual encodings, e.g. aggregated numerosity in Figures 2a and 2c, and contour density polygons in Figure 2b.

To this end, we build a virtual hierarchical tree of objects as follows. Each zoom level constructed in Phase I forms one level of the tree. Nodes of the tree are objects. For each object on level i, use its nearest neighbor on level $i-1$ as its ancestor.

With this tree, we can compute aggregation metrics, e.g., average/std/quantile of another field, convex hulls and top-k items, recursively from the bottom level up. Note that this tree does not have to be materialized. We maintain an R-tree for each level for fast ad-hoc nearest neighbor queries. The following example illustrates the process to calculate aggregated numerosity.

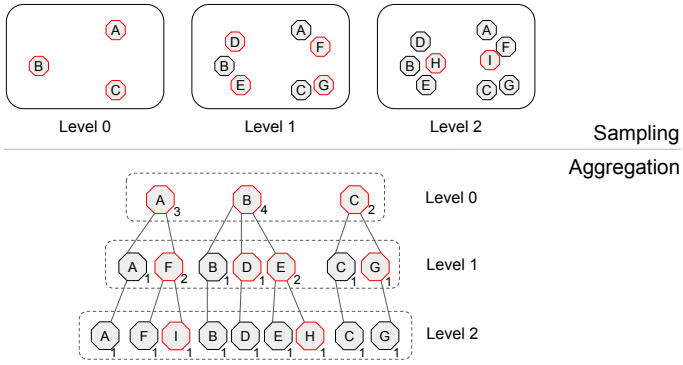


Figure 3. An illustration of Phase I and Phase II. There are 9 objects in total (A-I). Objects A-C are sampled on level 0. Objects D-G are sampled on level 1. Objects H and I are sampled on level 2. The number next to the objects in the tree represents how many objects an object represents, i.e., aggregated numerosity as in Figures 2a and 2c.

Note that the contour density polygons in Figure 2b are also generated using the aggregated numerosity in Figure 3 to approximate kernel density estimations. More specifically, instead of computing KDE values using all objects, we use the numerosity count as the weight of a sampled object. In other words, every object is represented by one sampled object. One big advantage of our approach is that every object is within a distance (bounding box size times θ) to the object that represents itself. As shown by [33], this leads to bounded KDE values.

4.3 Complexity Analysis

Both Phase I and Phase II have a time complexity of $O(n \log n)$. Phase I uses an R-tree of existing samples to perform efficient overlap test, while Phase II builds an R-tree for each zoom level for fast nearest neighbor queries. Currently, AutoDD has a single-node implementation of Phase I and Phase II. We plan to investigate parallelization opportunities in order to fully fulfill **G2-b**.

5 Conclusion

In this paper, we presented the design of AutoDD, its current implementation and future plans. AutoDD employed a declarative model that allowed easy specification of ZSV design choices from a large design space. A two-phase sampling and aggregation algorithm was devised to construct a hierarchy of zoom levels to automatically manage occlusion and clutter. AutoDD had been integrated with Kyrix to achieve interactive response times on big datasets.

REFERENCES

[1] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1363–1375. ACM, 2016.

[2] B. Bederson and J. Hollan. Pad++: a zooming graphical interface for exploring alternate interface physics. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 17–26. ACM, 1994.

[3] B. Bederson, J. Meyer, and L. Good. Jazz: an extensible zoomable user interface graphics toolkit in java. In *The Craft of Information Visualization*, pages 95–104. Elsevier, 2003.

[4] C. Beilschmidt, T. Fober, M. Mattig, and B. Seeger. A linear-time algorithm for the aggregation and visualization of big spatial point data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 73. ACM, 2017.

[5] H. Chen, W. Chen, H. Mei, Z. Liu, K. Zhou, W. Chen, W. Gu, and K.-L. Ma. Visual abstraction and exploration of multi-class scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1683–1692, 2014.

[6] H. Chen, S. Engle, A. Joshi, E. D. Ragan, B. F. Yuksel, and L. Harrison. Using animation to alleviate overdraw in multiclass scatterplot matrices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 417. ACM, 2018.

[7] P. Crickard III. *Leaflet. js essentials*. Packt Publishing Ltd, 2014.

[8] A. Das Sarma, H. Lee, H. Gonzalez, J. Madhavan, and A. Halevy. Efficient spatial sampling of large geographical tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 193–204. ACM, 2012.

[9] N. Elmqvist and J.-D. Fekete. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):439–454, 2009.

[10] J.-D. Fekete and C. Plaisant. Interactive information visualization of a million items. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, pages 117–124. IEEE, 2002.

[11] T. Guo, K. Feng, G. Cong, and Z. Bao. Efficient selection of geospatial data on maps for interactive and visualized exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 567–582. ACM, 2018.

[12] F. Heimerl, C.-C. Chang, A. Sarikaya, and M. Gleicher. Visual designs for binned aggregation of multi-class scatterplots. *arXiv preprint arXiv:1810.02445*, 2018.

[13] J. Jo, F. Vernier, P. Dragicevic, and J.-D. Fekete. A declarative rendering model for multiclass density maps. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):470–480, 2018.

[14] D. A. Keim and A. Herrmann. *The gridfit algorithm: An efficient and effective approach to visualizing large amounts of spatial data*. IEEE, 1998.

[15] R. Kosara, S. Miksch, and H. Hauser. Focus+ context taken literally. *IEEE Computer Graphics and Applications*, 22(1):22–29, 2002.

[16] H. Liao, Y. Wu, L. Chen, and W. Chen. Cluster-based visual abstraction for multivariate scatterplots. *IEEE transactions on visualization and computer graphics*, 24(9):2531–2545, 2017.

[17] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, 2013.

[18] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

[19] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[20] A. Mayorga and M. Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE transactions on visualization and computer graphics*, 19(9):1526–1538, 2013.

[21] F. Miranda, L. Lins, J. T. Klosowski, and C. T. Silva. Topkube: a rank-aware data cube for real-time exploration of spatiotemporal data. *IEEE Transactions on visualization and computer graphics*, 24(3):1394–1407, 2017.

[22] D. Moritz, B. Howe, and J. Heer. Falcon: Balancing interactive latency and resolution sensitivity for scalable linked visualizations. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, page 694. ACM, 2019.

[23] C. A. Pahins, S. A. Stephens, C. Scheidegger, and J. L. Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE transactions on visualization and computer graphics*, 23(1):671–680, 2016.

[24] D. Park, S. M. Drucker, R. Fernandez, and N. Elmqvist. Atom: A grammar for unit visualizations. *IEEE transactions on visualization and computer graphics*, 24(12):3032–3043, 2017.

[25] A. Perrot, R. Bourqui, N. Hanusse, F. Lalanne, and D. Auber. Large interactive visualization of density functions on big data infrastructure. In *2015 IEEE 5th Symposium on large Data Analysis and Visualization (lDAV)*, pages 99–106. IEEE, 2015.

[26] E. Pietriga. A toolkit for addressing hci issues in visual language environments. In *null*, pages 145–152. IEEE, 2005.

[27] A. Sarikaya and M. Gleicher. Scatterplots: Tasks, data, and designs. *IEEE transactions on visualization and computer graphics*, 24(1):402–412, 2017.

[28] W. Tao, X. Liu, Y. Wang, L. Battle, Ç. Demiralp, R. Chang, and M. Stonebraker. Kyrix: Interactive pan/zoom visualizations at scale. In *Computer Graphics Forum*, volume 38, pages 529–540. Wiley Online Library, 2019.

[29] M. Trutschl, G. Grinstein, and U. Cvek. Intelligently resolving point occlusion. In *IEEE Symposium on Information Visualization 2003 (IEEE Cat. No. 03TH8714)*, pages 131–136. IEEE, 2003.

[30] C. Waldeck and D. Balfanz. Mobile liquid 2d scatter space (ml2dss). In *Proceedings. Eighth International Conference on Information Visualisation, 2004. IV 2004.*, pages 494–498. IEEE, 2004.

[31] L. Wang, R. Christensen, F. Li, and K. Yi. Spatial online sampling and aggregation. *Proceedings of the VLDB Endowment*, 9(3):84–95, 2015.

[32] A. Woodruff, J. Landay, and M. Stonebraker. Constant density visualizations of non-uniform distributions of data. In *ACM Symposium on User Interface Software and Technology*, pages 19–28, 1998.

[33] Y. Zheng, J. Jestes, J. M. Phillips, and F. Li. Quality and efficiency for kernel density estimates in large data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 433–444. ACM, 2013.