EFFECTIVE DENSITY VISUALIZATION OF MULTIPLE OVERLAPPING AXIS-ALIGNED OBJECTS

NILOY COSTA

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

JANUARY 2020

Abstract

Large-scale analytics of multiple overlapping axis-aligned objects is a challenging computational geometry problem that can inform several applications and services, in diverse domains. The primary focus of this research is, given many axis-aligned objects, to devise efficient and effective data visualization methods that inform *whether*, *where* and *how much* they overlap. Currently, such visualizations rely on inefficient implementations to determine the size of the overlap of objects. We address this problem by exploiting state-of-the-art computational geometry methods based on the sweep line paradigm. These methods are fast and can determine the exact size of the overlap of multiple axis-aligned objects, therefore can effectively inform the visualization method. Towards that end, we propose OL-HeatMap, a novel density-based visualization technique that can be used to represent complex information about overlapping objects. Our experimental evaluation demonstrates the effectiveness of the proposed method in several synthetic and real-world data sets.

Acknowledgements

I want to take this opportunity and express my gratitude towards Professor Manos Papagelis for his continuous support and belief in me. Without his supervision, guidance and constant push, I would not have been able to finish my work. Thank you.

I want to thank Professor Enamul Hoque Prince for making me a part of his research and providing guidance in Data Visualization since day one.

I wish to sincerely thank my thesis advisory committee member Professor Scott MacKenzie and Professor Marin Litou for their time and valuable suggestions and comments to improve the quality of my thesis.

I want to thank York University, the Faculty of Graduate Studies, and Lassonde School of Engineering for their support and all the opportunities they have provided. Thank you York University Graduate Student Association and Electrical Engineering and Computer Science Graduate Student Association for letting me be a part of the team and providing me with leadership opportunities.

I would like to express my profound gratitude to my beloved parents, for their love and encouragement. My father have always encouraged me to be the best version of myself, and my mother have provided the mental support throughout the entirety of my Masters' Journey. I am especially grateful to my sister Meghla Costa for being the most understanding person and helping me believe that my dream is achievable.

Finally, I would like to express my very sincere thanks to my friends, Tilemachos Pechlivanoglou, Sherry Innocent, Mehedi Hasan Pranggon and Wenxiao Fu for their support. Without their presence and influence, this thesis would not have been possible.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Density-based Data Visualization Methods

There are various data analysis problems, where the data points are distributed across some space, and we are interested in retrieving information related to the underlying properties of that distribution. They provide powerful abstract representations of large data sets that can help one to quickly perceive areas of interest due to a large concentration of data points (or their absence). One of the most common ways to get some insight about this kind of problem is to visualize it using a density-based visualization technique [27].

1.1.1 Heat-maps

Amongst a plethora of visualization techniques for density, such as scatter plots, 2-D density plots or treemaps, we focus on one of the most commonly used density-based visualization methods, the *heat-map*. A heat-map is a graphical representation of data where data values are represented as colors. These colors depict the characteristics of the data based on problem-specific requirements. Typically, darker colours depict regions with higher amounts or concentrations of data values present, while the opposite is true for lighter colors. Sequential colors are used to represent data values in this case. These scales point to which values are larger / smaller and the distance between two values [45]. Variants of heat-maps

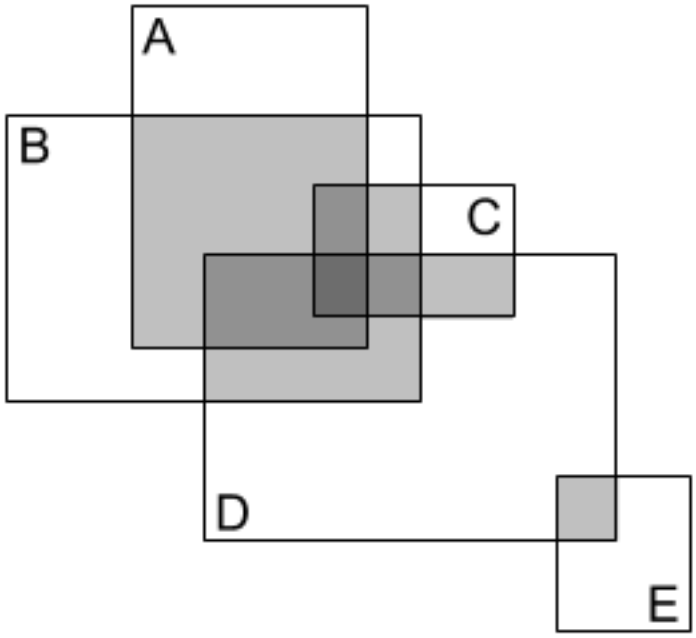
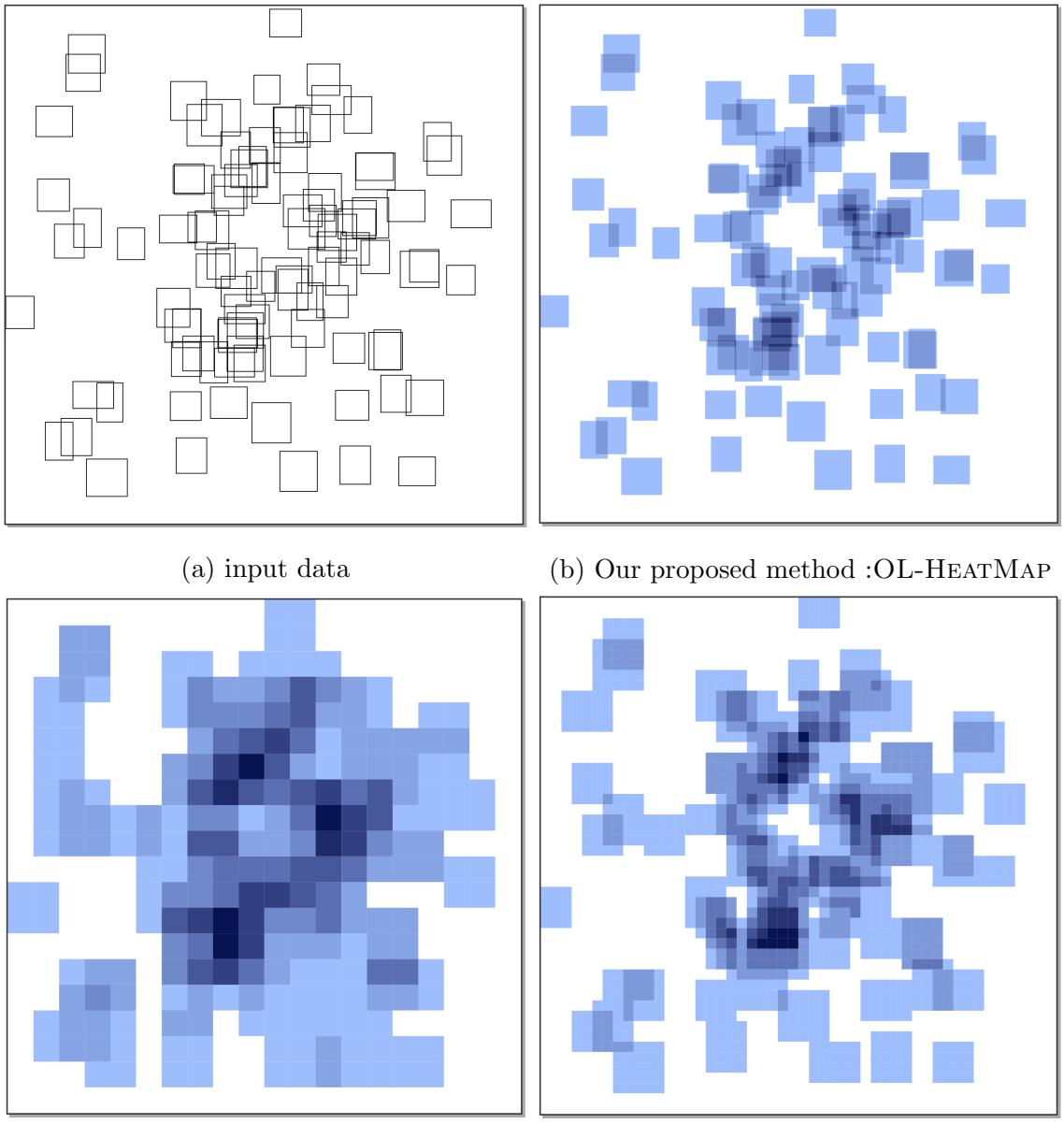


Figure 1.1: Illustrative example of a few overlapping rectangles. We propose effective methods (fast and exact) for a density-based visualization of the multiple overlaps.



(a) input data

(b) Our proposed method :OL-HeatMap

(c) grid-based [20x20]

(d) grid-based [50x50]

Figure 1.2: Input data and rectangle density visualizations.

have been used to show the density or distribution of data on a given region of interest. Heat-maps is a versatile visualization technique as it can suit to different types of data-sets, such as 1D [17] and 3D [26] as well. Essentially, this technique provides a general view of numerical data, and it can be customized to support statistical and categorical data variants. It can also be employed to show the results of clustering algorithms. As the rendered graphic is easy to understand, it is typically used to check the expected results versus the actual results of an algorithm.

1.1.2 Bounding Boxes

A common tool employed in the construction of a heat-map visualization is related to *bounding volumes*, and specifically *bounding boxes*. A bounding volume is a visual abstraction that is used to approximate complex objects and simplify the visualization process. Such visual abstractions introduce some flexibility to the problem, allowing for faster computation while avoiding significant losses in the information visualized. For different objects in real life, different bounding volumes such as rectangles, cuboids, spheres, and hyper-planes can be used. Furthermore, when the shapes used are rectangles or cuboids, they can be *axis-aligned*, meaning that their sides are parallel to the respective coordinate axis. In this work, we focus on *2-dimensional axis-aligned bounding boxes* (or *rectangles*). Previously, such bounding boxes have been used to approximate geographical objects [25], for the construction of spatial data structures [31], but also in VLSI design [14], to name a few.

1.1.3 Naive Approach and Limitations

We are interested in creating density-based visualizations that offer insights about the interactions (relationships) of these bounding boxes on a Cartesian plane. To that end, we need to identify and report the density value (i.e., the number of rectangles that overlap) of every point on the Cartesian plane. In addition, for each of these overlaps we want to determine the size of the overlap and its location in the Cartesian plane. There are a handful of approaches to address this problem, with one of the most common being grid-based [15]. According

to grid-based methods, first a uniform grid is defined that would separate the observation space into equal size grid cells. Then, the method determines the overlap of each grid cell to the input rectangles using well-established orthogonal range query methods [30], such as *r-trees* [18]. However, grid-based methods inherit several limitations. Constructing a spatial grid-based data structure and performing range queries for each grid cell is computationally expensive. Furthermore, the accuracy of the visualization results would greatly depend on the size of the grid (grid granularity). This presents an interesting trade-off where a small grid will be computationally more efficient but less accurate, and a large grid will provide more accurate representation of the overlaps, but at the expense of running cost. An illustrative example of this trade-off is shown in Fig. 1.2. We further elaborate on this trade-off in the methodology and experimental evaluation sections.

A more desirable outcome would be to be able to identify the *exact location*, *density* and *size* of any overlap among the available rectangles in the data-set directly. The simplest, brute-force approach to accomplish this is to compare every rectangle with every other rectangle, pair-wise first, then proceed to compare the overlap of every pair with every other object to find triple overlaps, and so on. As is apparent, the computational cost of such a method is prohibitively high. Instead, an approach that is commonly used to answer such geometric object overlap problems efficiently is the algorithmic paradigm known as the *sweep-line* or *plane sweep* algorithm [36]. Algorithms belonging to this category utilize a conceptual line that sweeps across the plane, quickly and efficiently identifying intersections between objects in the process.

1.2 Research Questions and Contributions

our research aims to answer the following research question.

1. What methods can be reconfigured to obtain an accurate representation of overlapping bounding boxes which are axis-aligned?

2. How to evaluate the efficiency of the methods for detecting multiple overlaps?

3. What would be the real-world use cases of our proposed approach?

In this work, we answer these question by employing a recently proposed variation of the sweep-line algorithm that is able to determine the exact location, size and number of multiple overlaps of n-dimensional geometric objects [32]. That method is using a sweep-line to construct an auxiliary data structure known as a *region intersection graph* and has the potential to significantly reduce the computation required for the effective visualization of the density of overlapping rectangles.

Specifically, the main contributions of our work are as follows:

1. We present OL-HeatMap (OverLap HeatMap), a fast and exact density-based visualization method for effective representation of the overlaps of multiple axis-aligned rectangles, based on the sweep-line paradigm.

2. We introduce an evaluation metric that can be used to determine the accuracy of grid-based heat-map visualizations. We perform a thorough experimental evaluation that provides evidence that OL-HeatMap is both faster and more accurate than competitive grid-based methods.

3. We implement OL-HeatMap for 1-Dimensional and 2-Dimensional data-sets, thus proving the versatility of our method and we demonstrate the scalability of OL-HeatMap and find intersections in high number of overlapping objects.

4. We build an interactive visualization dashboard that demonstrates the effectiveness of OL-HeatMap in practice. We visualize real world data-sets using OL-HeatMap and show the impact of our methodology in several use cases.

5. We make *source code* and *data* publicly available to encourage reproducibility of method and results.

1.3 Thesis Organization

The thesis have been organized in the following way. Chapter 2 discusses the previous research done on density-based visualizations and computing overlaps. We then proceed to provide an introduction to formalize the problem of interest in Chapter 3. Our proposed methodology as well as the traditional methods are discussed in Chapter 4. Chapter 5 presents a thorough experimental evaluation of the methods and algorithms. Chapter 6 presents an overview of our demo dashboard system, along with real world use cases of our proposed methodology. We conclude in Chapter 7 with future directions of our research.

Chapter 2

Related Work

The work in this thesis is related to *density-based visualization methods* and methods for computing *rectangle overlaps*. A number of key ideas on finding density and methods to visualize them have been discussed in this section. The methods for detecting overlaps have been discussed in this section as well. These areas have already been mentioned throughout this paper, and here we present a more comprehensive view of existing work on these topics.

2.1 Density-based Visualization Methods

A density-based visualization is adequate for noticing changes in the data, visualizing clusters and pointing out outliers [44]. Due to their utility, several methods have been proposed over the years for density-based representations. Visualization techniques such as 2D Scatterplots, Histograms, Time Series plots can be considered as techniques which provide an individual marker for each data point. However, with the highly dense data-sets, these methods become obsolete as they are unable to provide any meaningful information from the visualized graphic [37]. As standard *scatterplots* started to become obsolete due to big data and over-plotting issues [5], de-cluttering methods started being proposed, including adding opacity [24], colour, smoothing [44] and/or binning the data. In many cases of density visualization, it can be noted how a combination of these aforementioned methods is being used. The

need for the aggregated markers, where a small area would contain aggregated information, would provide more screen real estate to be visualized. Density plots, Treemap are examples of visualization of aggregated data.

2.1.1 Kernel Density Estimation and Histograms

The classical method of density visualization is to use Kernel Density Estimation [38] and histogram; these are used to find the distribution of a random variable. The Gaussian kernel can be expressed using the following equation -

$$KDE(x) = \frac{1}{n}\sum_{i=1}^{n} w_i f_\sigma(dist(x, x_i)) \quad f_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

This is used to generate a continuous function from a discrete point set. Perrot et al. describes the equation as "a point of weight w spreads a weight $wf\sigma(d)$ for any location at distance d. The bigger σ is, the further a point will spread its weight.[34]" KDE, along with different variants such as AKDE(Approximate KDE) [34] and SKDE(Super KDE) is used to calculate levels of abstraction from the data-set. The problem with KDE is, the computation needs to be repeated for every pixel to visualize. So if there are n number of data points with p number of pixels, the complexity would be *O(n * p)*. The complexity creates a scalability issue as we are stepping into the age of big data.

Kernel Density Estimation (KDE), along with its variants, such as Approximate KDE (AKDE) [34] and Super KDE (SKDE) are used to calculate levels of abstraction from the data-set. The problem with KDE is that the computation needs to be repeated for every pixel visualized. Therefore, if there are n number of data points with p number of pixels, the complexity would be $O(n \cdot p)$, which means it cannot scale well with very large data. Methods such as Curve Density Estimates [20] have also been proposed that use a KDE-based operation for rendering smoothed data. Histograms can perform similar operations at a lower computational cost but introduce higher opportunity costs. For instance, using a histogram to calculate a density distribution means the outcome will be less smooth. Histograms are also limited in high dimension and there are constraints on sub-bandwidth [21]. As large

data-sets cannot be loaded into the main memory to perform relevant exploratory operations, some pre-processing methods have been proposed to facilitate density-based visualizations.

2.1.2 Sampling

Sampling reduces the data size as it removes data points. This method is applicable when there is less variance in the data-set as it can remove interesting data points as well. Chen et al. [7] discusses the advantages and disadvantages of using an adaptive hierarchical multi-class sampling technique to visualize multi-class scatter-plots while the features of the data-set are preserved.

Data classification as a part of preprocessing is one of the methods which is also used to optimize scatterplots [29]. As scatterplots fail due to complexity when data points increase, based on analysis tasks, there are various design approach which can be undertaken for a successful visualization. Stochastic or stratified sub-sampling are methods used in different use cases [7] [4]. Methods related to sampling essentially reduces the dataset before it can be mapped to a visual representation method.

2.1.3 Simplifying Visual Representation

There are strategies which address overplotting by simplifying the visual representation itself. Consider the space for creating the graph to be a real estate, the issue has been addressed by a combination of visualization techniques. Starting with Bubble Sets [9], where it is done by adding visualization layers on top of the existing representations. The visualization technique used there is Contour Plots. Landscape maps [41] has also been used to achieve the same goal. In this case, the key idea is to use height to describe the density of that data point. Splatterplots [28] addresses the overdrawing of data points by using data abstraction to bound the density of data and adding a zoom functionality to achieve accurate information retrieval.

2.1.4 Clustering

Clustering provides the opportunity to merge data points which eventually reduces data points. For density-based visualizations, clustering is repeated to create a hierarchical data structure, such as a cluster tree [1, 10]. An *antichain* can be selected which serves as an abstraction. This abstraction can be used to make the visualization interactive as well.

A combined approach of sampling data points from clusters provide the best of both strategies. In the case of subsampling [16], the goal is to create an uniform grid-based representation of the original data-set. Uniform grids have a distinct advantage in having a smaller size when it comes to creating the grids itself. These strategies are important as they are space-optimized and provide flexibility for fast and local data exploration.

2.1.5 Binned Aggregation

Binned Aggregation is also used as a data reduction technique instead of sampling as the latter tends to remove data points completely. The cost of KDE can be avoided by using binned aggregation. Liu et al. proposed $ImMens$ [23] which groups data points into pre-defined "bins". These "bins" are not dependent on other "bins" which ensures that parallel computing can be used too. Li et al. [21] used KDE to binned data points for creating a multilevel heat-map. A combination of a binned aggregation approach with KDE-based methodologies has been used to visualize dense time series data [44] [20].

2.1.6 Big Data Processing Frameworks

Big data architectures are used to scale up to larger data-sets. For large spatio-temporal data-sets, a specialized system [12] has been proposed to process data and render images of heat-maps. The MapReduce framework has also been used [43] to distribute existing algorithms. Perrot et al. discussed using the Apache Spark framework to perform canopy clustering in order to create low-latency heat-maps [35].

Software Tools such as Tableau, Platfora, Datameer Analytics Solution and Cloudera

provide flexibility in using big data processing frameworks such as Haddop ecosystem [40].

2.2 Computation of Rectangle Overlaps

Several data mining and knowledge discovery problems can be modeled and solved by reducing them to the rectangle overlap problem, where we seek to find information about the overlapping behavior of a large number of rectangles in a data-set. Speed and accuracy is the key when it comes to detect collisions or intersections of rectangles. In this section, we discuss a few methods that have been proposed to approach the problem.

2.2.1 Sweep-line based Methods

Shamos and Hoey proposed the sweep-line algorithm, reducing the complexity of the naive approach of detecting intersections of n elements with a more efficient complexity of $O(n \cdot logn)$ from the naive $O(n^2)$ [36]. For a 1-dimensional approach, sweep-line based algorithms tend to perform better than other methods. One of the methods that have been common in detecting overlaps is to use the 1D approach on a 2D plane at the beginning, which is to run the conceptual sweep line and then use brute force on each pair of intersections to test for intervals. A tree-based data structure, such as interval tree, can be used simultaneously for better performance [8]. However, for denser data-sets, this method is computationally expensive. Sweep line has also found application in graph mining, for instance in the problem of finding important nodes in trajectory networks[33].

Plane-sweep is a go-to method for detecting overlaps as it is temporally optimized. Edelsbruner [11] provides an optimized plane-sweep algorithm which provides the flexibility to be expanded into more dimensions.

2.2.2 Division into Sub-spaces

A plane that contains all the rectangles/bounding boxes can be subdivided into grids or cells and then individual operations can take place for each cells. The idea of a uniform

grid has been proposed to detect collisions [15], which provides an opportunity for parallel computation as well. However, the problem with this approach is that the accuracy of finding overlaps or creating correct visual abstraction depends entirely on the cell size (or grid granularity). Another drawback of using a uniform grid is that getting the right grid size is a process of trial and error. Van Hook et al. proposed a 2^d data structure for dynamic adjusting of the grid cell size [42]. In spite of these drawbacks, Uniform Grid remains a cheaper alternative [16] than other methods for visualizing density, calculating intersections due to the cheaper cost in creating and storing the grid itself.

2.2.3 Partition-based Data Structure

The key idea of this method is to recursively insert rectangles into the root of a tree-based data structure. Bounding Volume Hierarchies have been proposed, such as Axis Aligned Bounding Boxes (AABB). Afterwards, the objects are tested in an iterative way against these data structures and inserted into the resulting tree. The $R-tree$ is one of the most discussed approaches to detecting overlaps [18]. $R-tree$ helps to group the objects into bounding rectangles of increasing size. Other methods include using a *range tree* based algorithms [3], which is not space-optimized. Usage of streaming algorithms [48] is another method proposed, but this method requires huge memory space to build and store the range tree. Tree based data structures provide an functionality to perform orthogonal queries for finding the intersections in optimized scenarios as well [46] .

Another key idea for visualizing volume based data is to use an adaptive scheme for rendering the volume itself into region-of-interest area and render the graphic in higher resolution [19]. This is achieved by sub-dividing the area using data structure such as $OctTree$. However, this particular data structure is typically used with another geometric object/s to have accurate computations[47]. For a large number of n objects, pair-wise intersections can be detected in real time by using conventional methods to detect collisions[22] using spatial and temporal coherence between successive instances. These methods work as the base of our work as we extend a sweep line based method to detect overlaps.

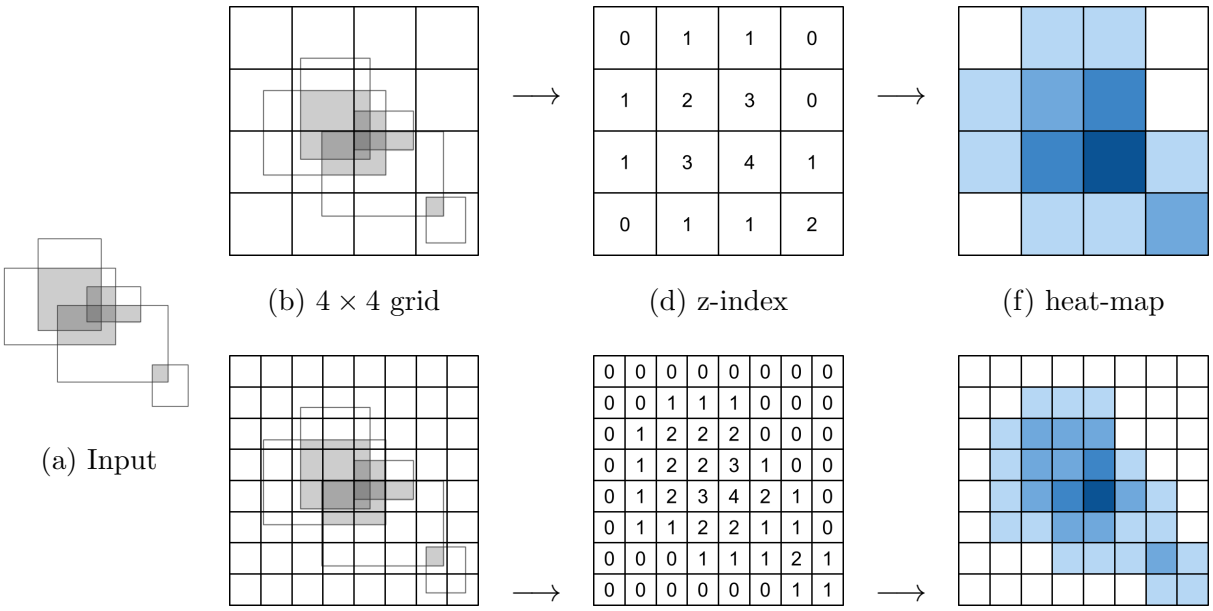
Table 3.1: Summary of Notations

Notation	**Description**
$\mathcal{R}$	A set of rectangles $\{R_1, R_2, \ldots, R_{n_R}\}$ in $\mathbb{R}^2$
n_R	Number of rectangles in $\mathcal{R}$
O_{AB}	Overlap of rectangles A and B
$S_{O_{AB}}$	Size of the overlap O_{AB}
O	A multiple-overlap of some rectangles in $\mathcal{R}$
S_O	Size of the overlap O
C	A set of grid cells: $\{C_1, C_2, \ldots, C_{g^2}\}$
g	Size of each grid side
(x, y)	XY-coordinates of a Cartesian point in $\mathbb{R}^2$
$z - index$	Number of rectangles a point (x, y) belongs to
k_{R_1}	Number of rectangles overlapping with rect. R_1

Chapter 3

The Problem

In this section, we introduce notation, provide preliminaries and formally define the problem of interest.



(a) Input (b) 4×4 grid (c) 8×8 grid (d) z-index (e) z-index (f) heat-map (g) heat-map

Figure 3.1: Grid construction and z-index calculation for different values of g.

3.1 Preliminaries

3.1.1 Rectangles

Consider the Cartesian plane $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$, where $\mathbb{R}$ is the set of all real numbers. Let a rectangle R be defined by the (x, y)-coordinates of two points in $\mathbb{R}^2$, one point representing its bottom-left corner (x_R^0, y_R^0) and one representing its top-right corner (x_R^1, y_R^1), respectively. As the two points represent the diagonal corners of the rectangle R, it is $x_R^0 < x_R^1$ and $y_R^0 < y_R^1$.

3.1.2 Pair-wise Overlapping Rectangles

Let a pair of rectangles A and B in $\mathbb{R}^2$ and the following two conditions:

$$max(x_A^0, x_B^0) \leq min(x_A^1, x_B^1) \tag{3.1}$$

$$max(y_A^0, y_A^1) \leq min(y_B^0, y_B^1) \tag{3.2}$$

The two rectangles A and B are intersecting if and only if both (3.1) and (3.2) are true. Note that (3.1), (3.2) check whether the rectangles are intersecting in the X-axis and Y-axis, respectively. When two rectangles A and B are intersecting, then their overlapping area defines a new rectangle, called an *overlap* and denoted as O_{AB}. The rectangle coordinates of O_{AB} are $(max(x_A^0, x_B^0), max(y_A^0, y_A^1))$ and $(min(x_A^1, x_B^1), min(y_B^0, y_B^1))$. Note that the dimensions of the overlap O_{AB} are given by:

$$width_{O_{AB}} = min(x_A^1, x_B^1) - max(x_A^0, x_B^0) \tag{3.3}$$

$$height_{O_{AB}} = min(y_B^0, y_B^1) - max(y_A^0, y_A^1) \tag{3.4}$$

The size $S_{O_{AB}}$ of the overlap O_{AB} is given by:

$$S_{O_{AB}} = width_{O_{AB}} \times height_{O_{AB}} \tag{3.5}$$

3.1.3 Multiple Overlapping Rectangles

Let $\mathcal{R} = \{R_1, R_2, ..., R_{n_R}\}$ be a set of rectangles in $\mathbb{R}^2$. In order to generalize the concept of *overlap* to more than two rectangles, we need to consider all the different ways that overlaps can occur. For example, in Figure 1.1, rectangles A, D and E have some pairwise overlaps (i.e., O_{AD} and O_{DE}), but they do not all overlap with each other forming a single *multiple-overlap* (O_{ADE}). On the other hand, for example, rectangles A, B, C, D are all overlapping with each other forming the multiple-overlap O_{ABCD}. Note that every point with (x, y)-coordinates that belongs to the rectangle defined by O_{ABCD} belongs to all four rectangles. Formally, for every point (x, y) of the observation space $\mathbb{R}^2$, we define its $z - index$. The $z - index$ refers to the number of distinct rectangles that the point with (x, y)-coordinates belongs to or otherwise it provides the number of multiple overlaps at each point of the observation space $\mathbb{R}^2$. Let O represent a *multiple-overlap* rectangle, and let (x_O^0, y_O^0) represent its bottom-left corner and (x_O^1, y_O^1) represent its top-right corner, respectively. Then, the size S_O of the *multiple-overlap* O is given by:

$$S_O = width_O \times height_O \tag{3.6}$$

where $width_O = x_O^1 - x_O^0$ and $height_O = y_O^1 - y_O^0$.

3.2 Problem Definition

We are now in position to formally define the problem of interest.

Problem 1 *Given a set of rectangles* $\mathcal{R} = \{R_1, R_2, ..., R_{n_R}\}$ *, where* R_i *is defined in an observation space* $\mathbb{R}^2$*, find the* $z - index$ *of every point with* (x, y)*-coordinates in* $\mathbb{R}^2$*.*

It is important to note here that all points of an area that represents a *multiple-overlap* will have the same $z - index$. So, the problem can be reduced to the problem of determining all *multiple-overlaps* O of a set of rectangles $\mathcal{R}$. Addressing this problem will provide information that can be utilized to create the desired density data visualization, including information about the $z - index$ of every point with (x, y)-coordinates in the observation space $\mathbb{R}^2$, along with information about the pair of points (x_O^0, y_O^0) and (x_O^1, y_O^1) that define each *multiple-overlap* rectangle O (one point representing its bottom-left corner and one representing its top-right corner, respectively), and the size S_O of the *multiple-overlap* rectangle O.

Chapter 4

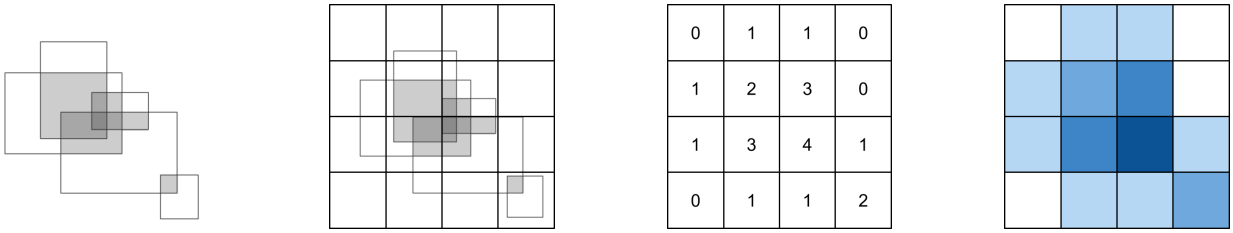
Methodology

In this section, we present the steps required for the visualization of bounding box heat-maps using the different approaches presented in previous sections. We start by describing the grid-based technique and the data structures necessary for its implementation. We proceed by outlining the basic sweep-line algorithm concept and specifically the multiple overlap identification and the intersection graph data structure required for it. Finally, we introduce an evaluation metric that can be used to determine the accuracy of the grid-based approach.

4.1 Grid-based Overlap Detection

Without loss of generality, for the remainder of this paper we will assume that all rectangles in the data-set are contained within a square Euclidean space R_l^2, where l is the size of the space in both dimensions (i.e. the length of square's sides). In the uniform grid-based approach, that space is divided into a set of $g \times g$ cells $C : \{C_1, C_2, \ldots, C_{g^2}\}$. This results in grid cells with equal size $\frac{l}{g} \times \frac{l}{g}$, and the parameter g determines the granularity of the grid. Figure 4.1 provides an overview of the grid based method.

In order to visualize the density of objects in the data, we seek to determine the number of rectangles that intersect with each cell (i.e., the cell's *z-index*), so that we can assign a



(a) input (b) 4x4 grid (c) z-index (d) visualization

Figure 4.1: Overview of the grid-based method.

relevant color to it. Since both the cell and the data objects are axis-aligned rectangles, this is an exact instance of the orthogonal range query problem. While it is possible to answer this problem in a brute-force way by comparing each cell with each rectangle in the data-set for overlap, this would be extremely time-intensive and inefficient. A number of well-established, state-of-the-art techniques exist; instead, that are specifically targeted towards providing a solution to this. Most of them employ tree-like data structures that allow for fast spatial queries, with one of the most common being *r-trees* [18]. In this approach, the tree is created by iteratively inserting input rectangles into it as leaf nodes, while the root node represents the entire space R_l^2 and the intermediate nodes represent groups of rectangles that lie within a minimum bounding box for each group. After the tree is constructed, fast search queries can be performed on it to identify all rectangles that intersect a given point or area by traversing the tree from the root to the leaf nodes. In conventional r-trees, the computation cost of the tree's construction is $O(n \log n)$, while the cost of each query is $O(\log n + k)$, where k is the number of intersecting pairs found.

For our problem of interest, the process mentioned above is followed and the r-tree data structure is constructed using all rectangles in the data-set. Afterwards, one query is performed for each cell to identify all the rectangles overlapping with the cell, and the count of retrieved results becomes the z-index value of the cell. To visualize the results, all that is necessary is drawing each grid cell using a color corresponding to its z-index value. This process is illustrated in Fig. 3.1 for different values of grid granularity g. As is apparent in the figure, higher granularity produces visualizations with greater accuracy, i.e., much closer

Algorithm 1: Grid-Based

Input: Set S of regions, set C of grid cells

Output: Set O of intersecting sets of regions, grouped by intersection cardinality k in each grid cells C in the form

$O = \{C_1 : [k_2 : [\{S_1, S_2\}, \ldots]], C_2 : [k_3 : [\{S_1, S_2, S_3\}, \ldots]], \ldots\}$

```
CreateGridCell(C)
O ← [ ]
CreateRTree(C,S)

for cell in Grid do
    SearchRTree(C)
    for i in [2,k-1] do
        O[i].append(all_combinations(cell, i))
```

to the original rectangle overlaps. However, as g increases, the number of grid cells (and therefore r-tree queries) increases quadratically. The process is explained in Algorithm 1.

4.1.1 Limitations

A worst-case scenario exists where all the rectangles in the data-set are overlapping at one or more grid cells, and therefore k can be very high; this, however, is a degenerate case for real-world scenarios and applications, as the heat-map visualization of such a case offers no actual meaningful information. Therefore, the total computation cost of the grid-based visualization is $O(n \log n + g^2(\log n + k_{C_i}))$. This often results in a situation where, depending on the value of granularity g selected, the end product is either significantly low-accuracy results or particularly slow execution times.

Procedure CreateGridCell(C)

```
C ← []
for x in Size do
    if regionset.dimension = 2 then
        for y in Size do
            new_cell = Region[lower[x0,y0],upper[x1,y1]]
        cells.append(new_cell)
    else
        new_cell = Region[lower,upper]
        cells.append(new_cell)
```

4.2 OL-HeatMap

A different approach to the problem involves identifying the *exact location, z-index* and *size* of any overlap among the available data-sets. This means that for every potential set of multiple overlapping rectangles, all the previous details of these overlaps need to be calculated. Once again, it is possible to answer this problem in a brute-force way by comparing every rectangle in the data-set with every other, finding overlapping pairs, and then proceed to compare the resulting overlaps with every other object to find overlapping triplets, and so on. As is apparent, the computational cost of such a method increases exponentially with the number of rectangles and quickly becomes unfeasible. Figure 4.3 provides an illustrative overview of the OL-HeatMap method. The overview of the Sweep Line algorithm is present in algorithm 2 and the Sweep Line Intersection Graph algorithm is present in algorithm 3.

Thankfully, better alternatives for addressing intersection problems exist in the literature. The most celebrated, state of the art methods for common intersection problems (e.g. finding pair-wise interval/rectangle intersections) are based on the *sweep-line* or otherwise *plane-sweep* algorithmic paradigm [2]. In this approach, a conceptual sweep line is used to identify and report intersections in Euclidean space. Given a set of 2-dimensional rectangles, the first

Procedure CreateRTree(R,C)

```
C ← [],R ← []
for i in Size do
    if regionset.dimension = 2 then
        x_min,y_min = Region.factor[lower(0,1)]
        x_max,y_max = Region.factor[upper(1,0)]
        index.insert(i, (x_min,y_min,x_max,y_max))
    else
        x_min,x_max = Region.factor[lower(0),upper(0)]
        index.insert(i, (x_min,x_max))
```

step of the algorithm involves constructing a list that includes the *left* and *right* X coordinates of all rectangles and sorting them, as a pre-processing phase. Then, the *conceptual line*, L moves (sweeps) from left to right across the plane, examining the rectangles, one by one, in order. During the sweep, the *active regions* (i.e., the ones that line L is currently traversing over) are maintained in a balanced tree structure. When L encounters a new region, its Y coordinates are compared with all the currently active regions to identify overlapping pairs, and the process completes after a single pass over the entire data-set. An illustrative example of the process can be seen in Figure 4.2a.

The process mentioned above identifies and reports all the pairs of overlapping rectangles in the data-set. However, to answer the problem of interest, it is necessary to find not only the pairs, but all multiple overlapping rectangles instead. A recently proposed variation of the sweep-line algorithm called SLIG can perform this task by utilizing a data structure known as the rectangle intersection graph [32]. A rectangle intersection graph is a graph where each vertex corresponds to a rectangle in the data-set and a connection between two vertices exists if and only if the respective rectangles are overlapping. In SLIG, the rectangle intersection graph is constructed during the sweep using the identified intersection pairs. A key concept for this intersection graph is the cliques. A k-clique in the intersection graph,

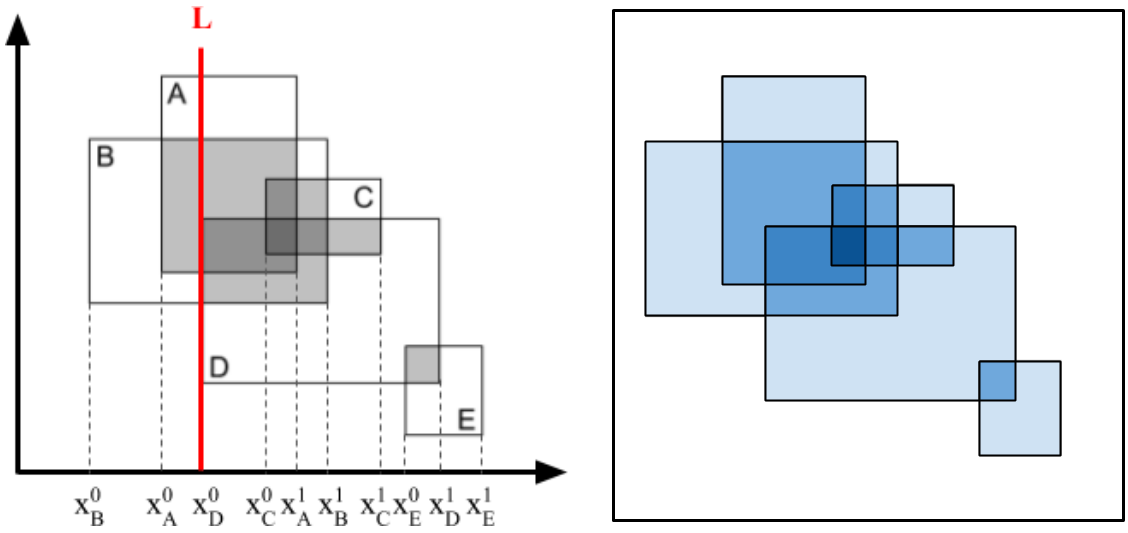
Procedure SearchRTree(*index*)

```
C ← []
R ← regionset

for cell in Grid do
    if regionset.dimension = 2 then
        overlap = list[index.intersection[
        [cell.factors[0].lower],[cell.factors[1].lower]],
        [cell.factors[0].upper],[cell.factors[1].upper]]]
    else
        overlap = list[index.intersection[
        [cell.factors[0].lower],[cell.factors[0].upper]],
```

corresponds to k objects that are simultaneously intersecting and share a common region. The problem of identifying multiple overlapping rectangles now becomes equivalent to that of enumerating all the possible cliques in the rectangle intersection graph, a well-studied problem in graph theory with several well-established state-of-the-art algorithms available [6] . The computation cost for this algorithm is $O(n \log n + n \cdot c_{max})$, where c_{max} is the number of maximal cliques found in the graph, or otherwise the number of unique sets of multiple overlapping rectangles that do not belong to more massive sets. The worst-case scenario for this algorithm is the same as the grid-based one, where all rectangles are overlapping at some point; this is also a degenerate for the reasons already mentioned.

We can now present OL-HeatMap, our approach to solving the rectangle heat-map visualization problem. Given a set or rectangles $\mathcal{R}$ as input, we first employ SLIG in order to identify all sets of multiple overlapping rectangles. Afterwards, we calculate the (x, y) coordinates and size S_O of each resulting overlap, along with its z-index value, which corresponds to the size of each set. Finally, to produce the heat-map visualization, all that is needed is to draw each of the resulting overlap rectangles with a color corresponding to the respective z-index, making sure that rectangles with higher z-index are in the foreground



(a) Sweep-line algorithm

(b) OL-HeatMap visualization

Figure 4.2: Illustrative example of a sweep-line algorithm and resulting OL-HeatMap visualization.

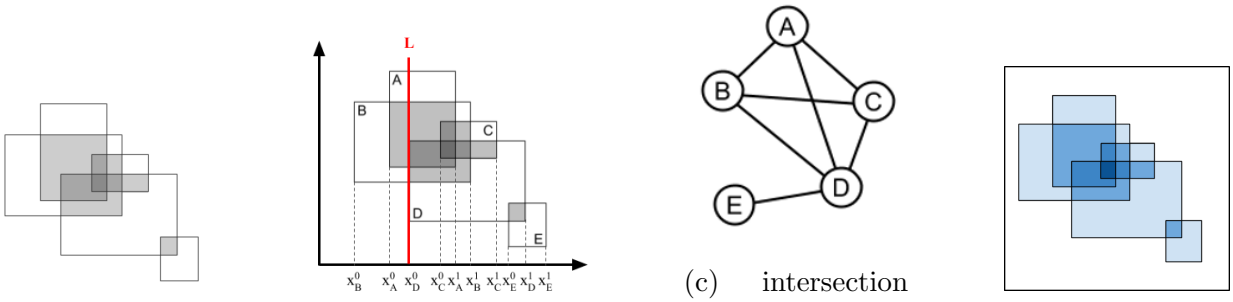
(i.e., are drawn last). An illustrative example of the resulting visualization is shown in Fig. 4.2b.

4.2.1 Advantages

As the OL-HeatMap method does not involve the use of a grid and therefore the computation cost multiplier g^2, it requires far less computation time compared to the grid-based approach for all but the smallest of grids. Furthermore, our approach calculates *exact* results, without any approximations. Therefore, to achieve visualizations with accuracy approach that of OL-HeatMap using a grid-based approach, the required grid sizes would be prohibitively large.

4.2.2 Limitations

The limitations of OL-HeatMap are minimal. As the Bron-Kerbosch clique enumeration algorithm is more efficient, there are degenerate cases where every regions are intersecting



(a) input (b) sweep-line (c) intersection graph (d) visualization

Figure 4.3: Overview of the OL-HeatMap method.

with each other, creating a very large high-clique intersection graph. Finding all the cliques in this graph would essentially increase the run-time. This case might occur in extremely dense data-sets but in real-life scenarios, there are not a lot of cases which are extremely dense. This behavior is due to an inherent limitation of the sweep-line method and this is a degenerate case, as most large data sets are typically very sparse. An experiment with a dense data-set generated using a gaussian distribution have been performed to check the run-time for finding all overlaps, and the limitations have been observed. On the other hand, streaming data-sets, where the data points are live, a traditional Sweep-Line algorithm will not be helpful as a sorted list of beginning and end-points of each regions need to exist before the conceptual sweep line can start from the left. OL-HeatMap is not suitable for data-sets in which we cannot use abstraction methods to approximate the objects. To sum, apart from the cases where the traditional sweep-line and clique enumeration algorithms have limitations, OL-HeatMap can perform faster with a 100% accuracy and it can be applied on any dimensions.

Algorithm 2: SweepLine

Input: Set S of regions

Output: Set O of intersecting sets of regions, grouped by intersection cardinality k in the form $O = \{k_2 : [\{s_1, s_2\}, \ldots], k_3 : [\{s_1, s_2, s_3\}, \ldots], \ldots\}$

Points $\leftarrow$ sort($x^0, x^1 \ \forall \ s_i, d \leftarrow 1$)

O $\leftarrow$ [ ], k $\leftarrow$ 2

LastIntersects $\leftarrow$ [[region] $\forall$ region in S]

while *O[k-1] not empty* **do**

    Intersects $\leftarrow$ GetKIntersects(k, LastIntersects)

    O[k] $\leftarrow$ Intersects

    LastIntersects $\leftarrow$ Intersects

    k $\leftarrow$ k + 1

Procedure GetKIntersects(k, LastIntersects)

```
Actives ← [], Intersects ← {}
for point in Points do
    if point.type = start then
        Intersects[point.region] ← []
        for activeIntersect in Actives do
            if activeIntersect.intersects(point.region) then
                intersection ← activeIntersect
                intersection.append(point.region)
                Intersects[point.region].append(intersection)
        for intersection in LastIntersects[point.regions] do
            Actives.append(intersection)
    else
        for intersection in LastIntersects[point.regions] do
            Actives.remove(intersection)
```

Algorithm 3: Sweep Line Intersection Graph

Input: Set S of regions

Output: Set O of intersecting sets of regions, grouped by intersection cardinality k in the form $O = \{k_2 : [\{S_1, S_2\}, \ldots], k_3 : [\{S_1, S_2, S_3\}, \ldots], \ldots\}$

Points $\leftarrow$ sort($x^0, x^1 \ \forall \ s_i, d \leftarrow 1$)

O $\leftarrow$ [ ]

GetIntersectionGraph(S)

GenerateKCliques(CliqueList,Graph)

for *clique in CliqueList* **do**

    k $\leftarrow$ len(clique)

    for *i in [2,k-1]* **do**

        O[i].append(all_combinations(clique, i))

Procedure GetIntersectionGraph(S)

Actives $\leftarrow$ [], Graph $\leftarrow$ []

for *point in Points* **do**

    if *point.type = start* **then**

        for *activeRegion in Actives* **do**

            if *activeRegion.intersects(point.region)* **then**

                Graph.addEdge(activeRegion, point.region)

        activeRegion.append(point.region)

    else

        activeRegion.remove(point.region)

Chapter 5

Experimental Evaluation

In this section we describe in detail the design and execution of the experimental evaluation of the different methods mentioned. Details on the data-sets used and the computational environment are provided and a comparison of performance and accuracy is presented for the OL-HeatMap and baseline uniform grid-based methods. We discuss the performance of OL-HeatMap from the perspectives of flexibility, versatility and scalability.

5.1 Environment

All experiments are conducted on a PC with 8x Intel(R) Core^{TM} i7-7700 CPU @ 3.60GHz and 64GB memory using Python 3.7. For each experiment, we execute the algorithm ten (10) independent times and report the average execution time or accuracy value.

5.2 Data

In order to evaluate the behavior of the algorithms under a wide variety of conditions, we make use of synthetic data. A data generator was implemented that produces data sets with specific characteristics thanks to a controlled number of parameters. For a square 2-D Euclidean space with side l, $n_{\mathcal{R}}$ rectangles were randomly generated with (x, y) coordinates

(a) uniform distribution

(b) triangular distribution

(c) gaussian distribution

(d) bi-modal distribution

Figure 5.1: Synthetically generated data-sets (examples).

Distributions \ Density	$n = 100$			$n = 500$			$n = 1000$			$n = 1500$			$n = 2000$		
	sparse	*dense*	*denser*	*sparse*	*dense*	*denser*	*sparse*	*dense*	*denser*	*sparse*	*dense*	*denser*	*sparse*	*dense*	*denser*
uniform	1	7	59	15	343	1272	42	5347	1314	112	28473	11866	208	5078	21065
triangular	3	20	93	16	616	2294	88	2397	10106	188	5520	22444	367	9256	39182
gaussian	5	78	479	95	2462	10317	415	10367	39976	881	22192	95458	1634	38349	164205
bi-modal	1	28	75	23	684	2851	112	2755	10712	216	5905	23390	405	11053	44216

Table 5.1: Summary of data-sets. The number reported for each data-set represents the *actual number of overlaps* found in it.

in $[0, l]$; unless otherwise noted $l = 1000$.

The size of each rectangle was randomly selected from the uniform range $[0, r \cdot l]$, where $r \in \{1\%, 5\%, 10\%\}$. Effectively, this means that the maximum length for the sides of the generated rectangles was a specific percentage of the total length of the space. As a result, the data-sets produced contain smaller or larger rectangles, which in turn means that there were fewer or more overlaps and the data-sets displayed lower or higher density, respectively. We refer to the data-sets produced with maximum length percentages $\{1\%, 5\%, 10\%\}$ as *sparse*, *dense* and *denser*, respectively.

The position of the lower x coordinate of each rectangle was randomly selected from one of four different probability distributions (uniform, triangular, gaussian, bi-modal). These distributions and their properties were selected to reflect a wide variety of possible real-world conditions; the gaussian distribution has mean value of $0.5l$ and sigma value of $0.2l$, while the bi-modal one is a combinations of two gaussians with mean values $0.2l$,$0.8l$ and sigma values $0.1l$,$0.1l$, respectively.

Therefore, the configurable parameters of the data generator are *number of objects* $n_{\mathcal{R}}$, *max length ratio* r and *spatial distribution*. For experimental evaluation purposes, various data-sets were created; their details are presented in Table 5.1.

5.3 Accuracy Evaluation Metrics

The baseline grid-based approach described in Section 4 does not provide exact results, but instead produces an approximation of the overlaps present in the data-set. As can be

intuitively understood from Fig. 3.1, a larger grid granularity value g produces a visualization that is closer to the actual overlaps that are present in the data-set. However, in order to thoroughly and objectively evaluate the effectiveness of the presented methods, it is necessary to utilize a definitive and unambiguous metric to quantify the accuracy of each visualization. To that end, we propose two (2) evaluation metrics that quantify the accuracy of grid-based visualizations of overlaps:

1) Percent of correct cells: A simple, straightforward way to determine the accuracy of a grid visualization is by only considering what percentage of the grid cells have a completely correct z-index value, and therefore the color. Cells that correspond to areas in the original data-set with the same z-index value at all their points (i.e., areas where nothing changes and no rectangle boundaries exist) are considered correctly visualized and all other cells are considered incorrect. Although this metric is easier to compute and simple to understand intuitively, it may not correctly reflect the accuracy of a visualization; if a grid cell's z-index value is the same for most, but not all of the corresponding areas in the data-set, the entire cell will be considered incorrect, while the actual error in visualization would be small.

2) Percent of correct area: A more refined and fair metric to evaluate the accuracy of a grid visualization is to consider what percentage of each cell's area correctly reflects the overlaps in the data-set. Each cell is compared to the area it corresponds to in the original data-set, and the extent of that area with the same z-index value as the cell is determined. Afterwards, this is used to calculate what percentage of that specific cell is correct or not, and the resulting percentages are averaged throughout the entire grid. Effectively, the value of this metric roughly corresponds to what percentage of the visualization has the correct z-index value (i.e., color).

As the second metric is considered more honest and refined, that will be used in the following experiments when reporting the accuracy of a grid-based visualization.

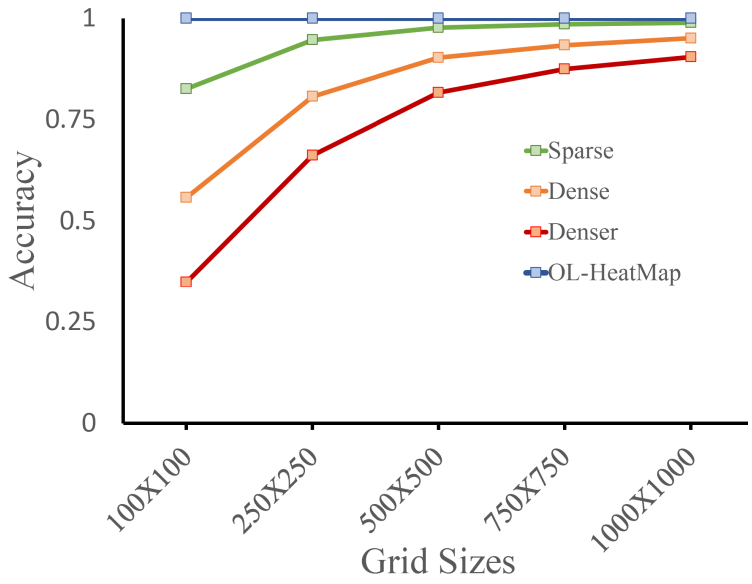
5.4 Experiments

We aim to evaluate the following aspects:

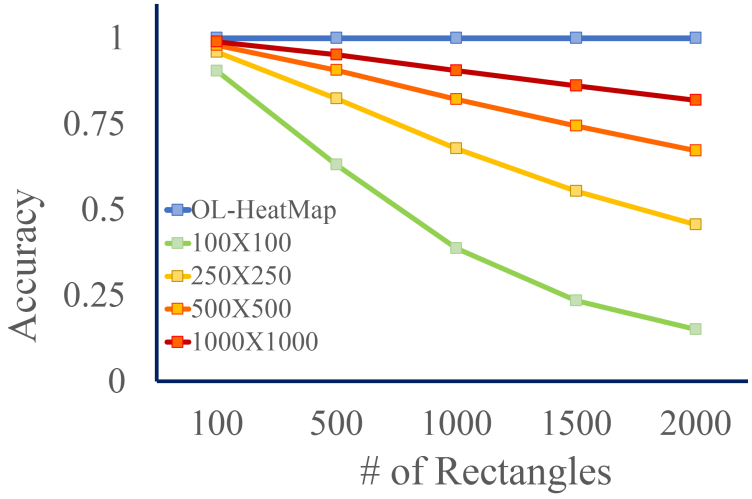
- OL-HeatMap **Accuracy Performance** How does grid size and granularity g affect the accuracy of the results of the grid-based method, for data-sets of different size and density?
- OL-HeatMap **Runtime Performance** How does our proposed method OL-HeatMap compare to the baseline grid-based method for the heat-map visualization problem in terms of execution time, for data-sets of different size and distribution?
- **OL-HeatMap Scalability** How does our proposed method OL-HeatMap scale for data-sets of larger sizes?
- **OL-HeatMap Versatility** How can we extend our proposed method OL-HeatMap to 1-D objects?
- **OL-HeatMap Flexibility** OL-HeatMap can be applied for the visualization of various real-world data-sets from various domains, as explained in Section 1. To demonstrate the its versatility, we apply OL-HeatMap to visualize two real-world data-sets.

5.4.1 OL-HeatMap Accuracy Performance

As mentioned preciously, the size and granularity of the grid can have significant impact on the accuracy of the resulting grid-based visualization. As the visualization that OL-HeatMap produces is always exactly correct, it is therefore of interest to examine what grid size and granularity values are required to achieve an accuracy that comes sufficiently close to the true results. To that end, we examined the visualization accuracy of the grid-based algorithm for data-sets of different densities, all selected from the uniform distribution and with the same number of rectangles $n_{\mathcal{R}} = 1000$. Furthermore, we measured the resulting accuracy for data-sets of different sizes, this time with the same density "dense" and once



(a) Measurement of accuracy for different grid sizes



(b) Measurement of accuracy for different data-set sizes

Figure 5.2: Accuracy performance of OL-HeatMap vs. grid-based

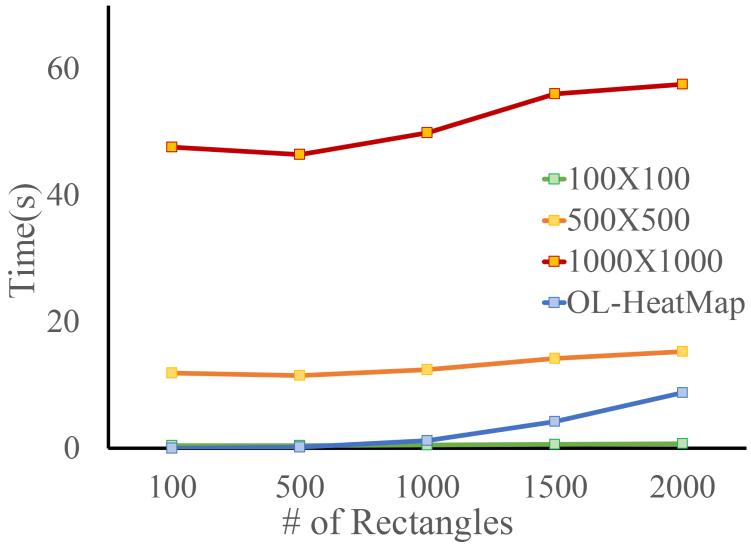
again uniformly distributed. The results for these experiments can be seen in Fig. 5.2a and Fig. 5.2b, respectively.

As expected, larger grids with higher granularity g result in more accurate visualizations. However, it can also be seen that larger or denser data-sets require accordingly large grids to achieve satisfactory results. This further highlights the value of OL-HeatMap, since it produces exact results that can be almost matched only by the largest of grids.

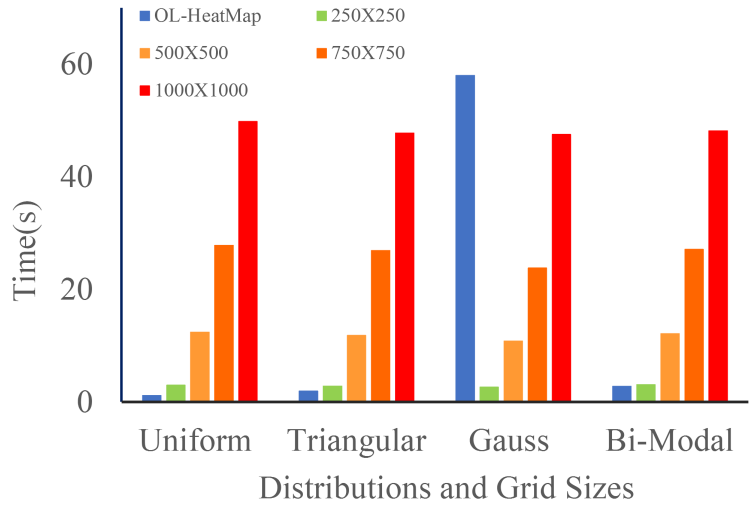
5.4.2 OL-HeatMap Runtime Performance

We evaluated time performance of OL-HeatMap against the baseline grid-based method, as a function of the number of rectangles $n_{\mathcal{R}}$ in the data-set. Furthermore, in order to explore a wide variety of scenarios and highlight the behavior of the two approaches for both convenient and unfavorable scenarios, we compared the time performance of the OL-HeatMap and grid-based approaches for the different distributions available. The spatial distribution of the rectangles is uniform in the first experiment, while the the number of rectangles $n_{\mathcal{R}}$ is fixed to 1000 in the second, while in both cases the data-sets are of "dense" density.

As can be seen in Fig. 5.3a and Fig. 5.3b, in most cases OL-HeatMap outperforms the baseline for all but the smallest (and therefore most inaccurate) grid sizes. A notable exception is when the objects follow a gaussian distribution; in that case, most objects are concentrated in the center of the space, producing a large number of overlaps and getting close to the worst possible scenario for the OL-HeatMap algorithm. Even in that case, however, the time performance of our approach is only slightly worse if not equivalent to the grid based approach for the largest grid granularity. It is also important to note that, as seen in the previous experiments, these high-density cases are the ones that require the largest grids to approach the accuracy of OL-HeatMap.



(a) Comparison of time for different data-set sizes



(b) Comparison of time for different data distributions

Figure 5.3: Time performance of OL-HeatMap vs. grid-based

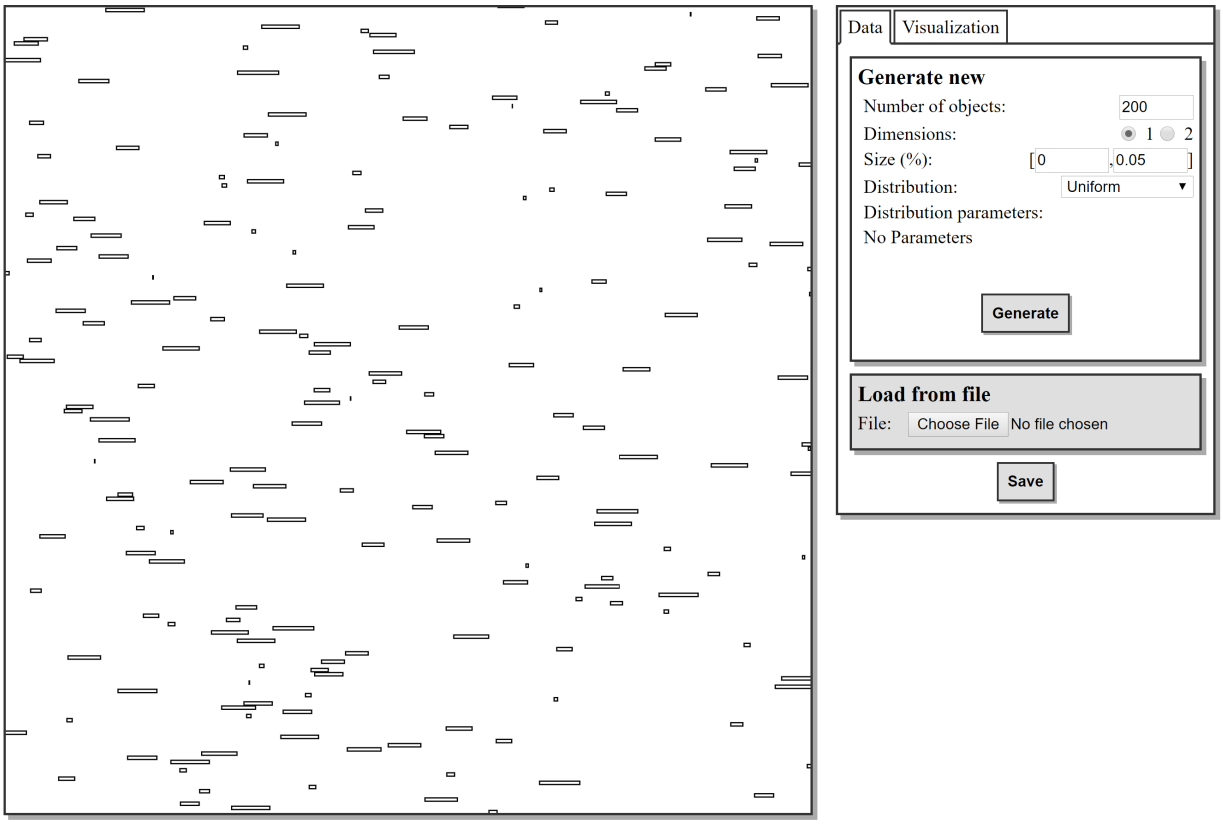


Figure 5.4: 1-D data generated, line segments depicting individual object

5.4.3 OL-HeatMap Versatility

We tested OL-HeatMap's versatility by generalizing our method to suit to 1-Dimensional data-sets. Considering 1-D as rectangles with a negligible yet fixed height, we can use OL-HeatMap to create regions which provide an 1-Dimensional view.

Data Generation

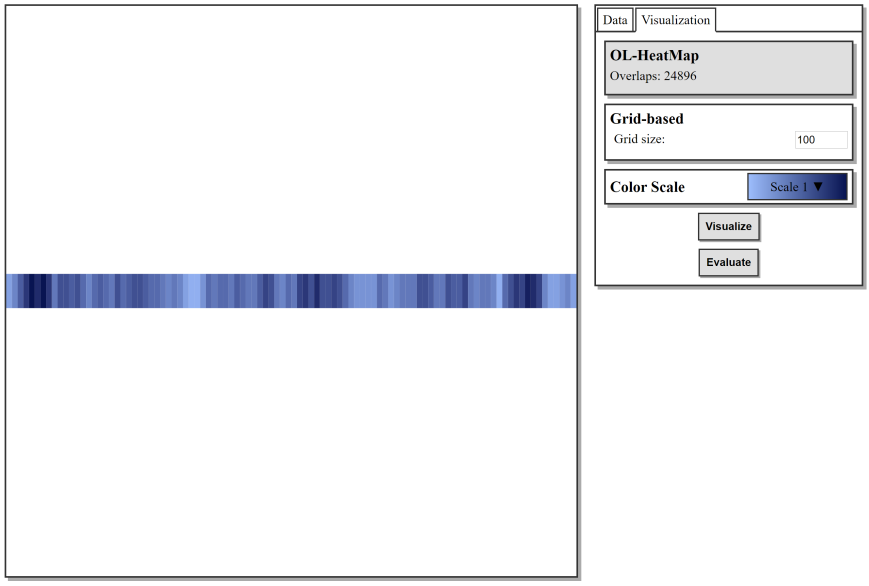
We consider the 1-Dimensional intervals as 2-Dimensional rectangles with a fixed height. As the baseline approach divides the area into grid cells for a 2-D approach, we generalized it for 1-Dimension by dividing the area into same-sized rectangles. Figure 5.4 provides a proof-of-concept view of the generated data-set. For the sake of simplicity, we have opted for a smaller size of data-set.

1-D Data Visualization

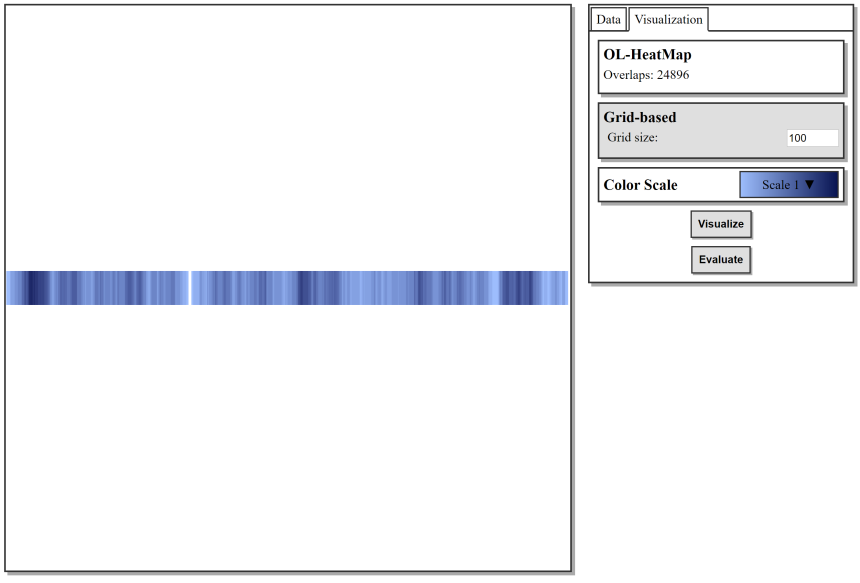
We create an 1-Dimensional visualization using D3. For OL-HeatMap, the visualization clearly shows change in overlap values. Figure 5.5 shows the Grid and OL-HeatMap of the generated data from Figure 5.4.

5.4.4 OL-HeatMap Flexibility

We used OL-HeatMap in real world scenarios. By applying OL-HeatMap in the two data-sets, which are, the Storm Event Database [13], and Airline On-time Performance Data [39], we have been able to find overlaps and retrieve density on a given area and then visualized. These different data-sets which have visualized using OL-HeatMap proves the flexibility of our method as we have generalized our approach to solve real world density visualization problem. Chapter 6 discusses these two data-sets, visualization goals and outcomes as well.



(a) 1-D visualization using grid



(b) 1-D visualization using OL-HeatMap

Figure 5.5: 1-D visualization using grid and OL-HeatMap

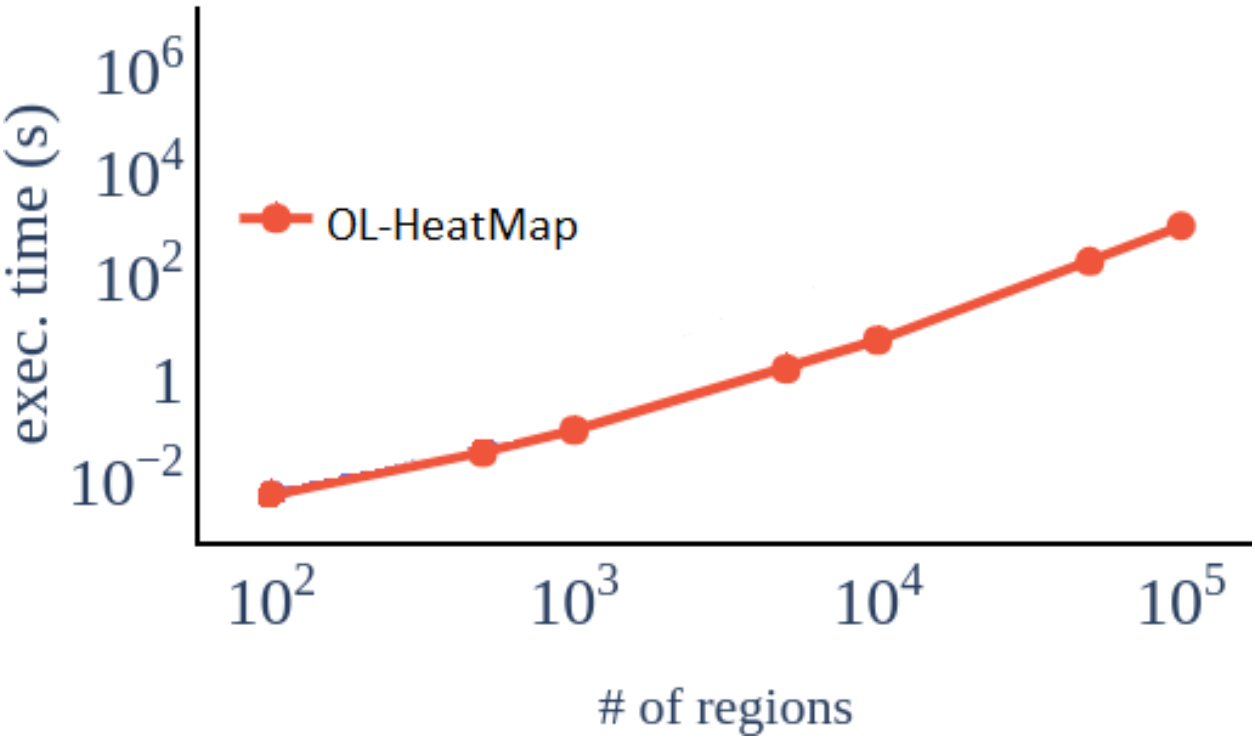


Figure 5.6: Scalability analysis

5.4.5 OL-HeatMap Scalability

We have conducted our comparative performance analysis for smaller data-sets, up-to a limit of 2000 overlapping rectangles. The reason behind this choice was to ensure quicker data visualization. However, irrelevant of the rendering methodology, OL-HeatMap can scale up to a high number of overlapping objects. Our goal for the scalability experiment was to find the amount of time it would take for OL-HeatMap to find the areas of multiple overlaps for a high number of 2D Rectangles.

Our method, OL-HeatMap can find $z-indexes$ and size of overlaps, S_O of overlapping objects faster while the number of objects increase. While, in the traditional uniform grid-based approach, finding intersections in a data-set of 10^5 rectangles is time consuming, OL-HeatMap can scale up to the task. Figure 5.6 shows that OL-HeatMap, can scale up-to 10^5 and more objects with relatively less amount of time. Interestingly, visualizing 10^5 overlapping objects with the HeatMap would be significantly challenging.

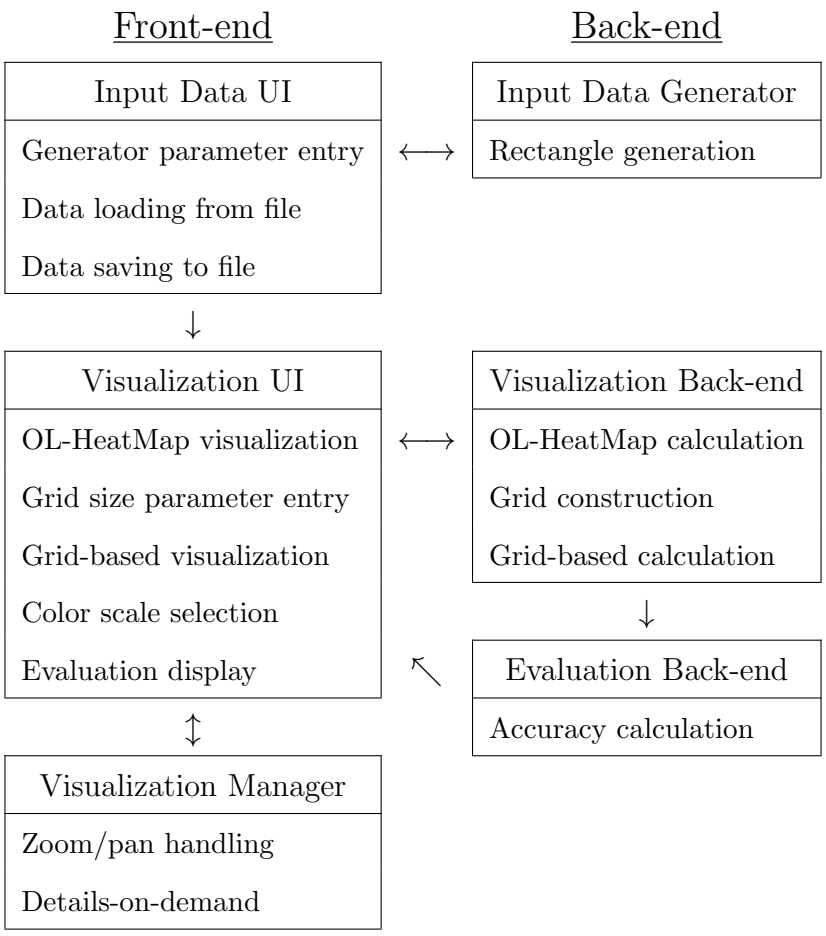
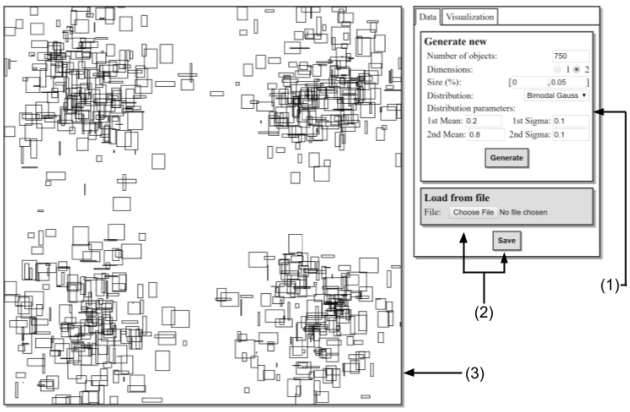
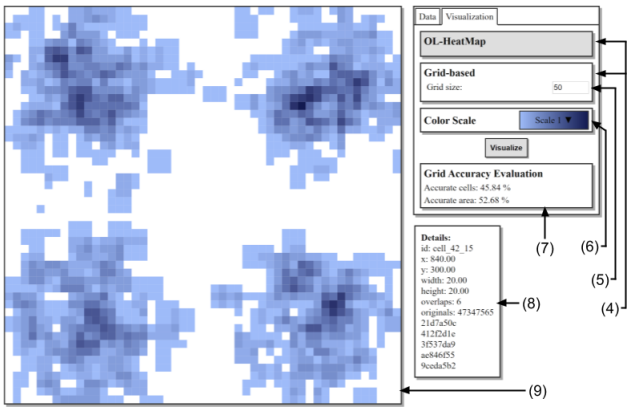


Figure 6.1: System architecture overview.



(a) Data input



(b) Grid-based visualization

Figure 6.2: The user interface (UI) of the demo. Highlighted features:

1. Random generator parameters
2. Data load/store operations
3. Data-set rectangles visualization
4. OL-HeatMap/grid-based selection
5. Grid granularity parameter
6. Color scale selection
7. Grid accuracy values
8. Rectangle/overlap/grid cell details
9. Overlap visualization

Chapter 6

Proof-of-Concept Demo System

In this chapter, we discuss the demo dashboard and three real world use cases of OL-HeatMap. We have designed our dashboard to have a client-server architecture which provides the functionality to effectively generate or load data-sets, find the overlaps of the bounding boxes and visualize accordingly. We use the Storm Event Database [13] and Airline On-Time Performance Data [39] to verify methodology.

6.1 System Architecture Overview

In this section, we present the architecture of the demo dashboard and its two distinct components: the front-end and the back-end. The front-end serves as a User Interface (UI), handling visualization and user interaction, while computationally intensive operations happen in the back-end asynchronously. Figure 6.1 provides an overview of the architecture.

6.1.1 Front-end

The front-end is responsible for the actual heat-map visualization and all interaction with the user. It is implemented in HTML, CSS and JavaScript, making use of the Data Driven Documents (D3) and JQuery JavaScript libraries for visualization and for communication with the back-end, plus other general functionality, respectively. The interface allows the

user to generate and store synthetic data-sets by specifying parameter values for the random data generator using form fields, or alternatively loading their own input data. The UI provides a preview of this input data, as well as the OL-HeatMap and grid-based heat-map visualization of overlaps in said data. The grid granularity value g, as well as the color scale used for the visualization can also be modified through form fields, while the visualization contains several useful features such as pan/zoom capability and details for each data point on hover. Finally, an evaluation of the grid-based approach's accuracy is displayed, both in terms of the percentage of cells that are correct (i.e., where every point in a cell has the correct overlap value) or the percentage of the total area that is correct (i.e., all points in the entire plot that have the correct overlap value). The user interface views for the data loading/generation and visualization can be seen in Figure 6.2.

6.1.2 Back-end

The dashboard demo is structured as a lightweight WebApp-style application, with a Python Flask-based back-end. The back-end contains the implementation of the data generator, as well as the OL-HeatMap and grid-based algorithms. For the grid-based algorithm, a grid is constructed over the input data-set according to the specified granularity value g, and each cell's overlap value is determined using an r-tree based index, with the help of Python's Rtree library. Likewise, the overlap rectangles for the OL-HeatMap method are calculated using the SLIG library. Finally, the results from the two methods are compared to produce the accuracy evaluation score for the grid-based approach, as the OL-HeatMap method by definition produces exactly correct results.

6.1.3 Visualization Limitations

We have designed and implemented our system to prove the correctness of our concept. This brings us to the limitations of this demo system. As this is a browser-based system at the moment, the performance of OL-HeatMap have limitations in terms of the drawing methodology and data loading. Previously, there were drawing limitations as we used D3 to

draw every single rectangle, albeit visible or not. However, we have been able to optimize our drawing function to skip drawing the rectangles which are completely covered by other rectangles. This optimization idea provides less amount of drawing needed than before. For example, in the case of Figure 6.4, while there are 130,651 overlaps present, only 94,621 of those are visible. So we draw 94,621 rectangles using D3 instead of 130,651 rectangles. As part of our future research, we intend to address the drawing problem completely and eliminate the choke-points in data to graphic rendering.

6.2 Real-world Use Cases

6.2.1 US Storm Event Data (2D)

Continental United States goes through a high number of natural calamities. Tornadoes are one the most recurring calamities, along with Flash Floods cause by higher rain, earthquakes caused by the San Andreas fault line. Tornadoes occur in the United States of America more than Europe. For example, entire Europe goes through around 300 Tornadoes every year, while United States of America faces around 1300 in that same year. There are various reasons as to why US gets more tornadoes, we aim to visualize these natural calamities effectively using OL-HeatMap.

The Data

The data-set we used in this case is the Storm Events Database[13] , this is an official publication of the National Oceanic and Atmospheric Administration (NOAA). It contains the documentation of each and every storm and other significant weather phenomena having sufficient intensity to cause loss of life, injuries, significant property damage, and/or disruption to commerce. The source includes National Weather Service (NWS) as their primary source of information, however, there are other sources contributing as well. The data-set contains all the relevant information regarding a significant weather phenomena, starting location, ending location, type of event (i.e, storm, tornado, flash flood, blizzards etc.), size

of the event, loss in terms of money and lives are present amongst other type of information. The data-set contains information from 1953 and it is updated regularly to have each event recorded. Events such as Hurricanes, Hail and Tornadoes have separate standards via which they are categorized. This ensures data consistency throughout the data-set as well as makes pre-processing easier.

Visualization Goal

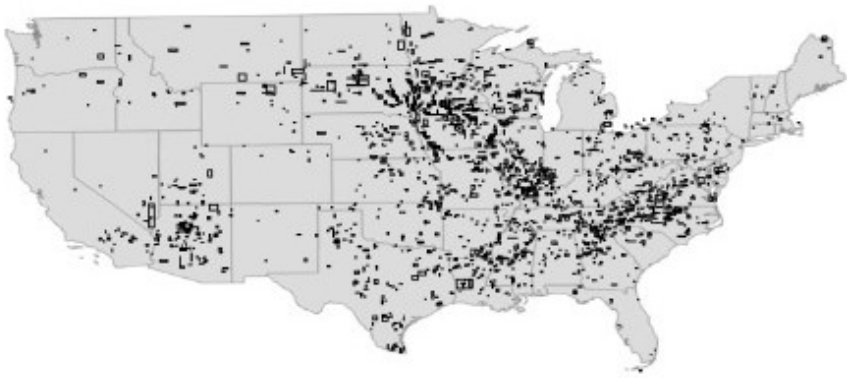
As a visualization goal, we want to ensure effective visualizations answering questions such as

1. Determining Storm Hot-spots in US
2. Effectively finding states with less severe weather incidents in the last 2 years
3. Finding all Storm events (Hurricanes) happened in one state (Florida) from 1953 - 2018
4. Visualize using OL-HeatMap to show severity of these storms

Visualizing All Storms in US for 2017 and 2018

Our first visualization goal indicates the need to create bounding boxes for each storm and then find their overlaps to effectively visualize all the storms happened in entire US. Towards this end, we use OL-HeatMap to find the answers to these questions. Firstly, we use convert our data-set in order to represent each event as a single bounding box. With all the events converted, we can use OL-HeatMap to find overlaps, and then visualize. Figure 6.3 provides an overview and a heat-map using OL-HeatMap of all the Tornadoes in United States for the year 2017 and 2018.

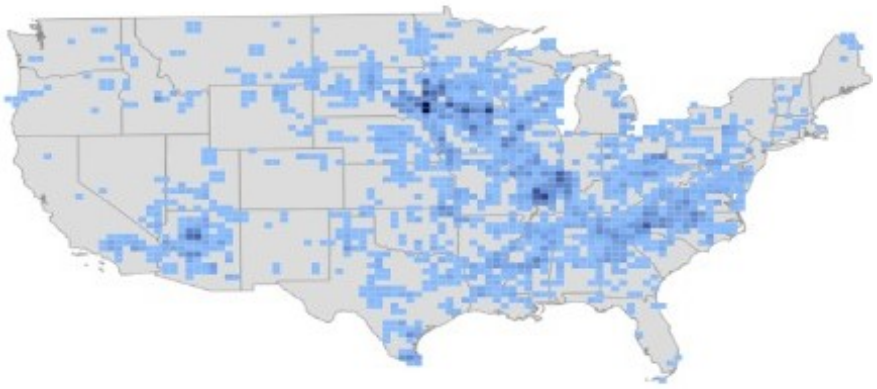
The interesting trend to note from these visualizations are the frequency of tornadoes in and around the East Coast states. This area, combined with Lower Mississippi valley has been aptly named as Dixie Alley or Tornado Alley, the highest concentration of Tornadoes in the US happens here.



(a) Data overview of storm events

(b) Data visualized using OL-HeatMap

Figure 6.3: Data overview OL-HeatMap visualization of all storms in US from 2017-18



(a) Grid based visualization

Overlap Counts

All Overlaps: 130651

Visible Overlaps: 94621

Grid Accuracy Evaluation

Accurate cells: 96.26 %

Accurate area: 96.35 %

(b) Accuracy evaluation of the grid based visualization

Figure 6.4: Grid based visualization of storms in the US during 2017-2018

On another hand, due to the position of Hawaii in the pacific ocean, there is a high number of tropical storms that happens in the season. Due to the amount of empty spaces in the map, the accuracy of the grid based method, as can be seen in Figure 6.4, is pretty high.

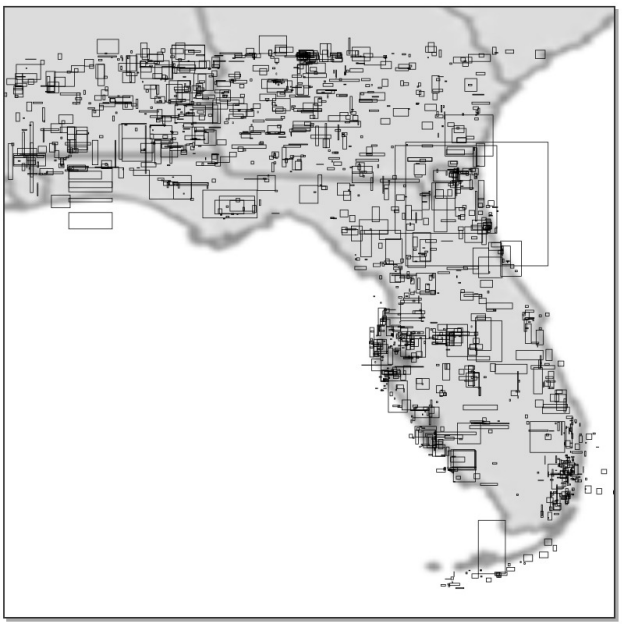
All Hurricanes in Florida

Every year Florida is ravaged by natural calamities. Amongst them, Hurricanes are the most common event recorded in Florida each year. The damage done by these hurricanes are catastrophic, an estimated 123 billion dollars worth of damage have been recorded from 2000 till to this date. As United States' densest hurricane zone, we visualized each hurricanes as their own bounding boxes and found overlaps and subsequently visualized using OL-HeatMap. The data contains all recorded hurricanes from 1953 to 2018.

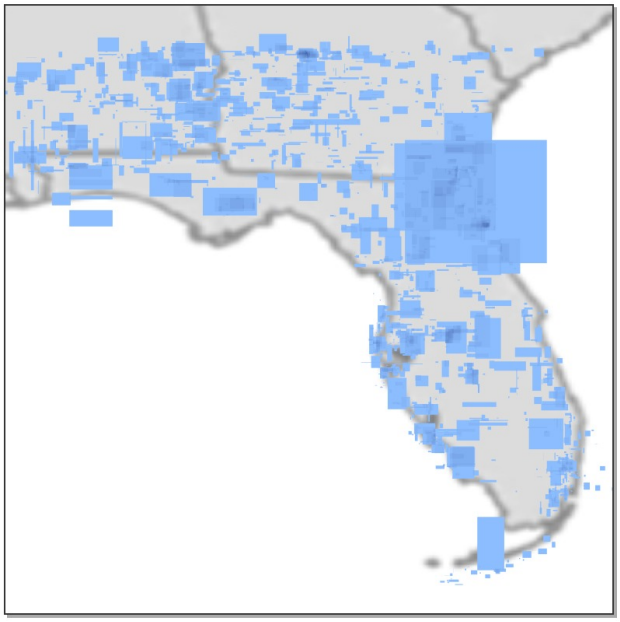
As can be seen from Figure 6.5, we determine the locations of our bounding boxes to be south of 82 Degrees Latitude. All Hurricane events which has a begin-latitude value of 82 have been counted in as an event hurling towards or originated in Florida. It cane be seen from the visualization that, almost all major cities in Florida have been under Hurricane attack since 1953, with the densest regions being Jacksonville, Fort Lauderdale, Miami, Daytona Beach, Port Orange and West Palm Beach. Figure 6.6 shows the grid based visualization of Florida, along with the all the visible overlaps with a 70 % accuracy, which indicates the amount of boxes which were drawn.

6.2.2 US Airline Carrier Data (1D)

The United States of America has 19,700 airports according to the 2011-2015 National Plan of Integrated Airport Systems (NPIAS). However, only 503 of these 19,700 airports serve commercial purposes, the data is available publicly for these 503 airports.

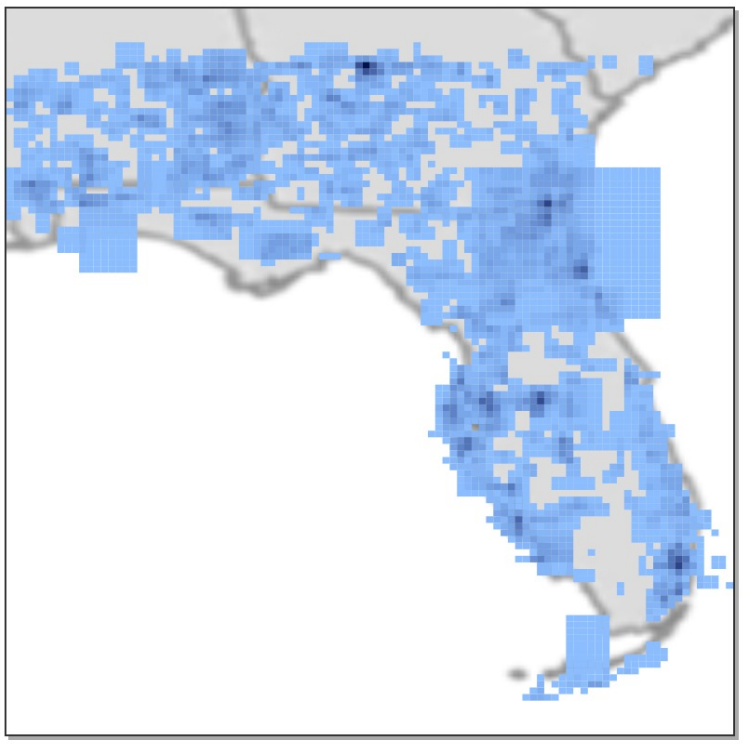


(a) Data overview of hurricane events



(b) Data visualized using OL-HeatMap

Figure 6.5: Data overview and visualization using OL-HeatMap of hurricane events in Florida from 1953-2018.



(a) Grid based visualization

Overlap Counts

All Overlaps: 43804

Visible Overlaps: 32774

Grid Accuracy Evaluation

Accurate cells: 71.37 %

Accurate area: 75.78 %

(b) Accuracy evaluation

Figure 6.6: Grid Based visualization of storms in the Florida during 1953-2019

The Data

BTS, also known as the Bureau of Transportation Statistics contains a plethora of publicly available data-sets, containing various information about various modes of transportation in United States. We used this public repository to find information about airline carriers. The data-set that we have used is Reporting Carrier On-Time Performance (1987-present), containing reports on-time data for flights they operate: on-time arrival and departure data for non-stop domestic flights by month and year, by carrier and by origin and destination airport. It also includes scheduled and actual departure and arrival times, canceled and diverted flights, taxi-out and taxi-in times, causes of delay and cancellation, air time, and non-stop distance. As a proof of concept, we analyzed a very short sub-set of the data itself, for John Wayne Airport, which is one of the least used commercial airports of the country. We visualized the traffic on one single day at the airport.

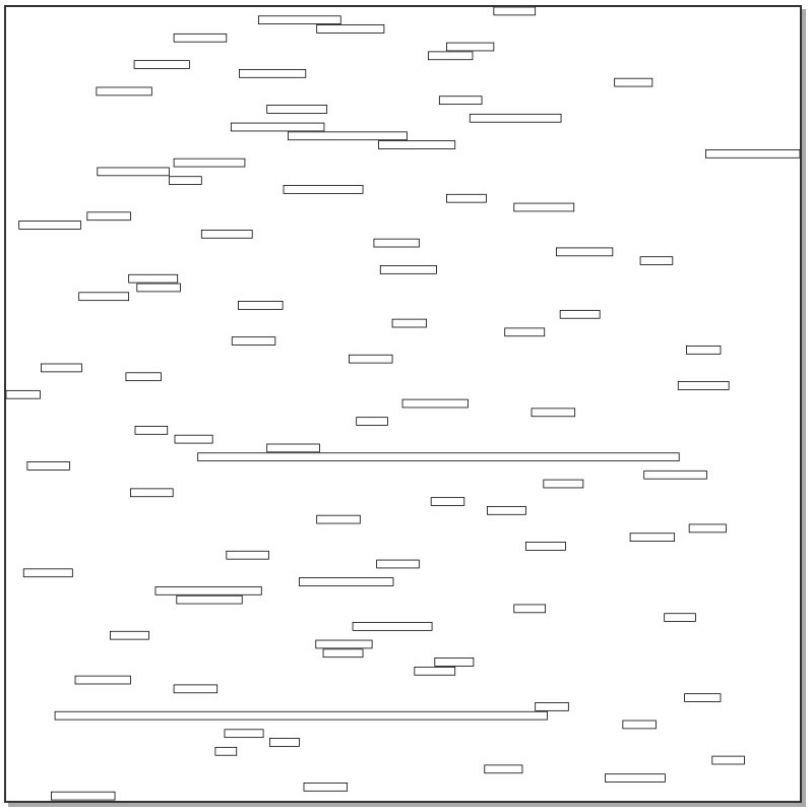
Visualization - Goal and Outcome

We fixed our visualization goal with as to find an optimized approach for visualizing airport usage by carriers. In order to aid in air and runway traffic management, the goals have been defined as the following -

1. Finding highest density of runway traffic
2. Finding least used time slot for a runway
3. Providing an overview of airport usage by airlines
4. Providing aid in Air Traffic Management

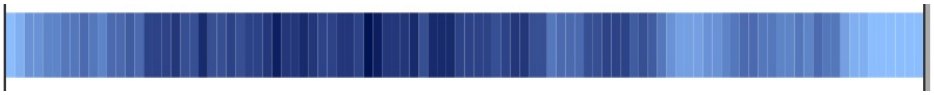
To achieve these goals, we visualized our data-set for February 1st, 2019 at the location John Wayne Airport, Orange County, California. Right off the bat from Figure 6.7, we have been able to notice the denser regions of the data-set, depicting the peak hour of runway traffic for this airport. We have also noticed the short amount of time each air-craft spends in this airport, which in comparison are different than a more commercially active airport

such as John F Kennedy International Airport (JFK) or Los Angeles International Airport (LAX) where air-crafts tend to spend longer hours after landing. This visualization can be of aid when balancing the load of adding new flights in this airport.



(a) Data overview

(b) 1-D visualization using OL-HeatMap. Time - 0000-2359 hours



(c) 1-D visualization using 100 grid. Time - 0000-2359 Hours [24 minute intervals]

(d) 1-D visualization using 50 grid. Time - 0000-2359 Hours [48 minute intervals]

Figure 6.7: 1-Dimensional data overview and visualization of JW Airport using OL-HeatMap and grid

Chapter 7

Conclusion and Future Work

7.1 Conclusions

Density-based visualizations, such as heat-maps, constitute a popular approach to visualize and perceive large amounts of complex data points effectively. In this research, we focused on a heat-map-like representation for the case of overlapping rectangles. This is a visualization problem that can guide powerful big data visual analytics and inform several applications in diverse domains. However, current state-of-the-art approaches to the problem rely on ad hoc naive implementations or methods that are known to not scale well, such as grid-based methods. Also, in order to perform reasonably fast, most of these methods provide approximations of the problem. To address these limitations, we have proposed OL-HeatMap, an effective method for finding and visualizing the exact density of overlapping rectangles, along with other useful information, including the actual position of the formed overlapping rectangle (*overlap*) and its size. Our method is based on a recently proposed variant of the sweep-line method that can accommodate multiple overlaps in n-dimensions. To demonstrate the effectiveness of OL-HeatMap against grid-based sensible baselines, we designed a thorough experimental evaluation incorporating various parameters and settings. Our proposed method is much more accurate as it always finds the **exact** solution and not an approximation of it. Furthermore, it performs **several orders of time faster** than its

competitors. An exception to this is the case of extremely dense data sets (i.e., almost all rectangles overlapping with each other), in which case OL-HeatMap can perform comparably to baselines; this behavior is due to an inherent limitation of the sweep-line method and this is a degenerate case, as most large data sets are typically very sparse. Overall, we expect OL-HeatMap to be integrated in information visualization software and libraries, as well as, to find application in several scenarios where visual analytics of large-size (potentially) overlapping axis-aligned objects is critical.

7.2 Future Work

OL-HeatMap provides accurate values for overlaps. We want to ensure better visualization stems out from our work. We have divided the future directions of our research into two distinct area of interest. The first one is to add more flexibility in our computations. The second research direction contains the relevant update to our drawing functions.

7.2.1 Computation

We want to include a weighted approach when we calculate overlaps. Ideally, we would be able to add weights for each intersections, thus the visualization technique can be generalized for more types of data in the future. A detailed and thorough evaluation testing the limit of the scalability of our method is also in the list of the future work. We intend to be able to easily find overlaps of a millions objects and then use a different method to visualize the overlaps. We would also use more data-sets to create effective heat-maps using the weighted approach.

7.2.2 Visualization

Efficient drawing of overlaps is one of the main draw-back of our current system. As we find higher amount of overlaps as we increase our input data, it becomes harder for our drawing function to effectively draw all of the boxes. We have already optimized by drawing the

least amount of boxes, there are room for optimizing our drawing function even further, by means of colouring each only once instead of repeating. For the time being, however, we have opted for a top-down approach to draw our visualization. In the future, we want to be able to draw each overlapped area once, thus saving time and resource to render visualized graphics.

Bibliography

[1] D. Auber and F. Jourdan. "Interactive refinement of multi-scale network clusterings". In: *Ninth International Conference on Information Visualisation (IV'05)*. London, UK: IEEE, July 2005, pp. 703–709. DOI: 10.1109/IV.2005.65.

[2] Bentley and Ottmann. "Algorithms for Reporting and Counting Geometric Intersections". In: *IEEE Transactions on Computers* C-28.9 (Sept. 1979), pp. 643–647. ISSN: 0018-9340.

[3] J. Bentley and D. Wood. "An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles". In: *IEEE Transactions on Computers* 29.07 (July 1980), pp. 571–577. ISSN: 1557-9956. DOI: 10.1109/TC.1980.1675628.

[4] Enrico Bertini and Giuseppe Santucci. "Give Chance a Chance: Modeling Density to Enhance Scatter Plot Quality Through Random Data Sampling". In: *Information Visualization* 5.2 (June 2006), pp. 95–110. ISSN: 1473-8716. DOI: 10.1057/palgrave.ivs.9500122. URL: http://dx.doi.org/10.1057/palgrave.ivs.9500122.

[5] D. B. Carr et al. "Scatterplot Matrix Techniques for Large N". In: *Journal of the American Statistical Association* 82.398 (1987), pp. 424–436. ISSN: 01621459. URL: http://www.jstor.org/stable/2289444.

[6] F. Cazals and C. Karande. "A note on the problem of reporting maximal cliques". In: *Theoretical Computer Science* 407.1 (2008), pp. 564 –568. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2008.05.010. URL: http://www.sciencedirect.com/science/article/pii/S0304397508003903.

[7] H. Chen et al. “Visual Abstraction and Exploration of Multi-class Scatterplots”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014), pp. 83–92. ISSN: 1077-2626. DOI: 10.1109/TVCG.2014.2346594.

[8] Jonathan D. Cohen et al. “I-COLLIDE: An Interactive and Exact Collision Detection System for Large-scale Environments”. In: *Proceedings of the 1995 Symposium on Interactive 3D Graphics*. I3D ’95. New York, NY, USA: ACM, 1995, 189–ff. ISBN: 0-89791-736-7. DOI: 10.1145/199404.199437. URL: http://doi.acm.org/10.1145/199404.199437.

[9] Christopher Collins, Gerald Penn, and Sheelagh Carpendale. “Bubble Sets: Revealing Set Relations with Isocontours over Existing Visualizations”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (Nov. 2009), pp. 1009–1016. ISSN: 1077-2626. DOI: 10.1109/TVCG.2009.122. URL: https://doi.org/10.1109/TVCG.2009.122.

[10] Jean-Yves Delort. “Vizualizing large spatial datasets in interactive maps”. In: *2010 Second International Conference on Advanced Geographic Information Systems, Applications, and Services*. IEEE. St. Maarten, Netherlands Antilles: IEEE, 2010, pp. 33–38.

[11] Herbert Edelsbrunner. “A new approach to rectangle intersections”. In: *International Journal of Computer Mathematics* 13.3-4 (1983), pp. 221–229. DOI: 10.1080/00207168308803365. eprint: https://doi.org/10.1080/00207168308803365. URL: https://doi.org/10.1080/00207168308803365.

[12] Ahmed Eldawy et al. “Shahed: A mapreduce-based system for querying and visualizing spatio-temporal satellite data”. In: *2015 IEEE 31st International Conference on Data Engineering*. IEEE. Seoul, South Korea: IEEE, 2015, pp. 1585–1596.

[13] National Centers for Environmental Information. *Storm Events Database*. 1953. URL: https://www.ncdc.noaa.gov/climate-information/extreme-events/us-tornado-climatology/trends.

[14] J. Fang et al. “A new fast constraint graph generation algorithm for VLSI layout compaction”. In: *1991., IEEE International Symposium on Circuits and Systems* (1991). DOI: 10.1109/iscas.1991.176140.

[15] Wm. Randolph Franklin et al. “Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines”. In: *Proceedings of Auto-Carto 9*. Baltimore, Maryland, US: Springer-Verlag, 1989, p. 100109.

[16] Lori A. Freitag and Raymond M. Loy. “Comparison of Remote Visualization Strategies for Interactive Exploration of Large Data Sets”. In: *Proceedings of the 15th International Parallel and Distributed Processing Symposium*. IPDPS 01. USA: IEEE Computer Society, 2001, p. 181. ISBN: 0769509908.

[17] Matthew Evan Garr, Jiri Rojicek, and Jiri Vass. *Heatmap timeline for visualization of time series data*. US Patent App. 13/086,255. Oct. 2012.

[18] Antonin Guttman. “R-trees: A Dynamic Index Structure for Spatial Searching”. In: *SIGMOD Rec.* 14.2 (June 1984), pp. 47–57. ISSN: 0163-5808. DOI: 10.1145/971697.602266. URL: http://doi.acm.org/10.1145/971697.602266.

[19] Eric LaMar, Bernd Hamann, and Kenneth I. Joy. “Multiresolution Techniques for Interactive Texture-based Volume Visualization”. In: *Proceedings of the Conference on Visualization ’99: Celebrating Ten Years*. VIS ’99. San Francisco, California, USA: IEEE Computer Society Press, 1999, pp. 355–361. ISBN: 0-7803-5897-X. URL: http://dl.acm.org/citation.cfm?id=319351.319432.

[20] O Daae Lampe and Helwig Hauser. “Curve density estimates”. In: *Computer Graphics Forum*. Vol. 30. Wiley Online Library. Llandudno, UK: The Eurographics Association and Blackwell Publishing Ltd, 2011, pp. 633–642.

[21] Chenhui Li, George Baciu, and Yu Han. “Interactive visualization of high density streaming points with heat-map”. In: *2014 International Conference on Smart Computing*. IEEE. Hong Kong, China: IEEE, 2014, pp. 145–149.

[22] M. C. Lin, D. Manoucha, and J. Canny. "Fast contact determination in dynamic environments". In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. May 1994, 602–608 vol.1. DOI: 10.1109/ROBOT.1994.351234.

[23] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. "imMens: Real-time Visual Querying of Big Data". In: *Computer Graphics Forum*. Vol. 32. Wiley Online Library. Leipzig, Germany: The Eurographics Association and John Wiley Sons Ltd, 2013, pp. 421–430.

[24] Justin Matejka, Fraser Anderson, and George Fitzmaurice. "Dynamic opacity optimization for scatter plots". In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM. Seoul, Republic of Korea: ACM, 2015, pp. 2707–2710.

[25] Takashi Matsuyama, Makoto Nagao, et al. "A file organization for geographic information systems based on spatial proximity". In: *Computer Vision, Graphics, and Image Processing* 26.3 (1984), pp. 303–318.

[26] Michael Maurus, Jan Hendrik Hammer, and Jürgen Beyerer. "Realistic heatmap visualization for interactive analysis of 3D gaze data". In: *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM. 2014, pp. 295–298.

[27] A. Mayorga and M. Gleicher. "Splatterplots: Overcoming Overdraw in Scatter Plots". In: *IEEE Transactions on Visualization and Computer Graphics* 19.9 (Sept. 2013), pp. 1526–1538. ISSN: 1077-2626. DOI: 10.1109/TVCG.2013.65.

[28] A. Mayorga and M. Gleicher. "Splatterplots: Overcoming Overdraw in Scatter Plots". In: *IEEE Transactions on Visualization and Computer Graphics* 19.9 (Sept. 2013), pp. 1526–1538. ISSN: 2160-9306. DOI: 10.1109/TVCG.2013.65.

[29] L. Micallef et al. "Towards Perceptual Optimization of the Visual Design of Scatterplots". In: *IEEE Transactions on Visualization and Computer Graphics* 23.6 (June 2017), pp. 1588–1599. ISSN: 2160-9306. DOI: 10.1109/TVCG.2017.2674978.

[30] Yakov Nekrich. "A linear space data structure for orthogonal range reporting and emptiness queries". In: *International Journal of Computational Geometry & Applications* 19.01 (2009), pp. 1–15. DOI: 10.1142/s0218195909002800.

[31] Dimitris Papadias and Yannis Theodoridis. "Spatial relations, minimum bounding rectangles, and spatial data structures". In: *International Journal of Geographical Information Science* 11.2 (1997), pp. 111–138.

[32] Tilemachos Pechlivanoglou, Vincent Chu, and Manos Papagelis. "Efficient Mining and Exploration of Multiple Axis-aligned Intersecting Objects". In: *Proceedings of the 19th IEEE International Conference on Data Mining*. IEEE. Piscataway, NJ, USA: IEEE, 2019, In Press.

[33] Tilemachos Pechlivanoglou and Manos Papagelis. "Fast and Accurate Mining of Node Importance in Trajectory Networks". In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 781–790.

[34] A. Perrot et al. "Large interactive visualization of density functions on big data infrastructure". In: *2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)*. Chicago, Illinois, USA: IEEE, Oct. 2015, pp. 99–106. DOI: 10.1109/LDAV.2015.7348077.

[35] A. Perrot et al. "HeatPipe: High Throughput, Low Latency Big Data Heatmap with Spark Streaming". In: *2017 21st International Conference Information Visualisation (IV)*. London,UK: IEEE, July 2017, pp. 66–71. DOI: 10.1109/iV.2017.45.

[36] Michael Ian Shamos and Dan Hoey. "Geometric intersection problems". In: *17th annual symposium on foundations of computer science*. Houston, Texas: IEEE, 1976, pp. 208–215.

[37] Ben Shneiderman. "Extreme Visualization: Squeezing a Billion Records into a Million Pixels". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD 08. New York, NY, USA: Association for Computing

Machinery, 2008, 312. ISBN: 9781605581026. DOI: 10.1145/1376616.1376618. URL: https://doi.org/10.1145/1376616.1376618.

[38] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[39] Bureau Of Transportation Statistics. *Airline On-Time Performance Data*. 1987. URL: https://www.transtats.bts.gov/DatabaseInfo.asp?DB_ID=120&Link=0.

[40] Venkataramana Sucharitha, SR Subash, and P Prakash. "Visualization of big data: its tools and challenges". In: *International Journal of Applied Engineering Research* 9.18 (2014), pp. 5277–5290.

[41] M. Tory et al. "Spatialization Design: Comparing Points and Landscapes". In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1262–1269. ISSN: 2160-9306. DOI: 10.1109/TVCG.2007.70596.

[42] Daniel J Van Hook, Steven J Rak, and James O Calvin. "Approaches to RTI implementation of HLA data distribution management services". In: *Proceedings of the 15th DIS Workshop*. 1996, pp. 535–544.

[43] Huy T Vo et al. "Parallel visualization on large clusters using MapReduce". In: *2011 IEEE Symposium on Large Data Analysis and Visualization*. IEEE. Providence, Rhode Island, USA: IEEE, 2011, pp. 81–88.

[44] Hadley Wickham. *Bin-summarise-smooth: a framework for visualising large data*. Tech. rep. University of Auckland, 2013.

[45] Claus O Wilke. *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. O'Reilly Media, 2019.

[46] Dan E. Willard. "New Data Structures for Orthogonal Range Queries". In: *SIAM J. Comput.* 14.1 (Feb. 1985), pp. 232–253. ISSN: 0097-5397. DOI: 10.1137/0214019. URL: http://dx.doi.org/10.1137/0214019.

[47] P. G. Xavier. “Fast swept-volume distance for robust collision detection”. In: *Proceedings of International Conference on Robotics and Automation*. Vol. 2. Apr. 1997, 1162–1169 vol.2. DOI: 10.1109/ROBOT.1997.614295.

[48] Afra Zomorodian and Herbert Edelsbrunner. “Fast software for box intersections”. In: *International Journal of Computational Geometry & Applications* 12.01n02 (2002), pp. 143–172.