Big Data Research 25 (2021) 100235

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

OL-HeatMap: Effective Density Visualization of Multiple Overlapping Rectangles

Niloy Eric Costa [*,1], Tilemachos Pechlivanoglou [*,1], Manos Papagelis

York University, 4700 Keele Street, Toronto, Ontario, Canada

ARTICLE INFO

Article history:
Received 1 October 2020
Received in revised form 11 February 2021
Accepted 11 April 2021
Available online 30 April 2021

Keywords:
Information visualization
Density-based visualization
Overlapping objects
Computational geometry
Heat-maps

ABSTRACT

Visualization of the density of multiple overlapping axis-aligned objects is a challenging computational problem that can inform large-scale visual analytics, in diverse domains. For example, when dealing with crowd simulations, we care about constructing interaction maps, and in urban planning we care about city areas mostly frequented by people, to name a few. The primary objective of this research is, given a large set of axis-aligned two-dimensional (2D) objects, or simply *rectangles*, to devise efficient and effective data visualization methods that inform *whether*, *where* and *how much* these rectangles overlap. Currently, such visualizations rely on inefficient implementations of determining the size of the overlapping rectangles that do not scale well and are hard to accomplish. Approximate methods have also been proposed in the literature. To the contrary of these approaches, we aim to 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 present OL-HeatMap, a novel type of a heat-map visualization that can be used to represent and perceive density of overlapping rectangles. Our experimental evaluation demonstrates the effectiveness of the proposed method in terms of both accuracy and running time for synthetic and real-world data-sets.

1. Introduction

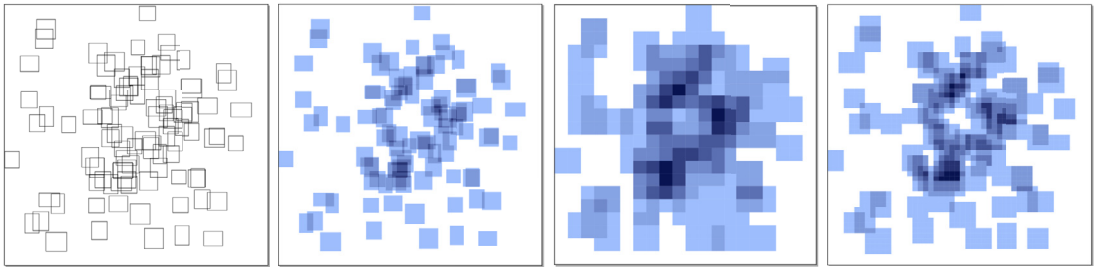
Density-based information visualization methods are commonly employed in big data visual analytics. They provide powerful abstract representations of large data sets that can help one to quickly perceive areas of interest due to a large concentration of data points (or their absence). Amongst a plethora of visualization techniques for density, such as scatter 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 colors depict regions with higher amounts or concentrations of data values present, while the opposite is true for lighter colors. Variants of heat-maps have been used to show the density or distribution of data on a given region of interest. This technique provides a general view of numerical data, and it can be customized to suit 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.

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* which we refer to as *rectangles* for ease of understanding. Previously, such rectangles have been used to approximate geographical objects [1], for the construction of spatial data structures [2], but also in VLSI design [3], to name a few.

We are interested in creating density-based visualizations that offer insights about the interactions (relationships) of these rectangles on a Cartesian plane. To that end, we need to identify and

* Corresponding authors.
E-mail addresses: ericnc@eecs.yorku.ca (N.E. Costa), tipech@eecs.yorku.ca (T. Pechlivanoglou), papaggel@eecs.yorku.ca (M. Papagelis).
[1] These authors contributed equally.

https://doi.org/10.1016/j.bdr.2021.100235
2214-5796/



(a) input data (b) OL-HeatMap (c) grid-based [20x20] (d) grid-based [50x50]

Fig. 1. Input data and rectangle density visualizations.

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 [4]. 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 [5], such as *R-trees* [6]. 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. 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 [7]. Algorithms belonging to this category utilize a conceptual line that sweeps across the plane and quickly identify overlapping objects.

In this work, we employ 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 [8]. 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:

- 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.
- We introduce an evaluation metric that can be used to determine the accuracy of grid-based heat-map visualizations.
- We conduct an extensive evaluation of the performance of OL-HeatMap which demonstrates that it significantly outperforms competitive grid-based methods, in terms of both running time and accuracy.

Table 1
Summary of notations.

Notation	Description
Ω	A 2-*dimensional* square observation space in $\mathbb{R}^2$
l	The length of the sides of Ω
$\mathscr{R}$	A set of n rectangles $\{R_1, R_2, \ldots, R_n\}$ in Ω
O	Set of multiple overlapping rectangles
$S_{O_{AB}}$	Surface area of the overlap O
C	A set of square grid cells: $\{C_1, C_2, \ldots, C_{g^2}\}$
g	Size of grid (nr. of rows or nr. of columns, since they are equal)
(x, y)	XY-coordinates of a Cartesian point in Ω
$z_{(x,y)}$	*z-index*, i.e. number of rectangles a point (x, y) belongs to
z_O	*z-index* value of multiple overlapping rectangles set O
z_{C_i}	*z-index* value of a grid cell
AC_{grid}	Percent of correct cells in a grid
AS_{grid}	Percent of correctly represented area inside the entire grid

- We build an interactive visualization system that demonstrates the effectiveness of OL-HeatMap in practice.
- We make *source code* and *data* publicly available to encourage reproducibility of method and results.[2]

The remainder of this paper is organized as follows: Section 2 introduces notation and formally defines the problem of interest in this paper. Our proposed method, OL-HeatMap, along with the grid-based competitors and the overall computational framework are presented in Section 3. Section 4 presents a thorough experimental evaluation of the methods and algorithms. After reviewing the related work in Section 6, we conclude in Section 7.

2. The problem

In this section, we introduce notation, provide preliminaries and formally define the problem of interest. A summary of all notations used are present in Table 1.

2.1. Preliminaries

2.1.1. Rectangles

Consider a 2-*dimensional* observation space Ω, which is a subspace of the Cartesian plane $\mathbb{R}^2$. Without loss of generality, for the remainder of this paper we assume that Ω is a square whose sides have length l, therefore $\Omega = \{(x, y) \in \mathbb{R}^2 \mid 0 \leq x \leq l \text{ and } 0 \leq y \leq l\}$. Let a rectangle R be defined by the (x, y)-coordinates of two points in Ω, 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 $y_R^0 < y_R^1$.

[2] https://github.com/ericnc09/OL-HeatMap.

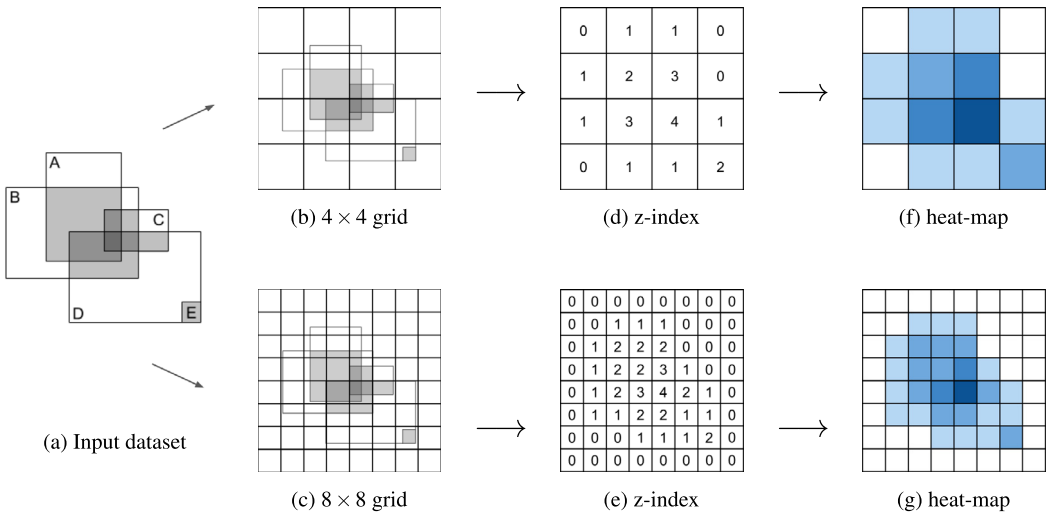


Fig. 2. Grid construction and z-index calculation for different values of g.

2.1.2. Pair-wise overlapping rectangles

Let a pair of rectangles A and B in Ω and the following two conditions:

$$max(x_A^0, x_B^0) \leq min(x_A^1, x_B^1) \quad (1)$$

$$max(y_A^0, y_A^1) \leq min(y_B^0, y_B^1) \quad (2)$$

The two rectangles A and B are intersecting if and only if both (1) and (2) are true. Note that (1), (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:

$$width_{O_{AB}} = min(x_A^1, x_B^1) - max(x_A^0, x_B^0) \quad (3)$$

$$height_{O_{AB}} = min(y_B^0, y_B^1) - max(y_A^0, y_A^1) \quad (4)$$

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

$$S_{O_{AB}} = width_{O_{AB}} \times height_{O_{AB}} \quad (5)$$

2.1.3. Multiple overlapping rectangles

Let $\mathscr{R} = \{R_1, R_2, ..., R_{n_R}\}$ be a set of rectangles in Ω. 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 Fig. 2a, 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 within the rectangle defined by O_{ABCD} belongs to all four rectangles. Formally, for every point (x, y) of the observation space Ω, we define its *z-index* value $z_{(x,y)}$. The *z-index* refers to the number of distinct rectangles that the point belongs to, or in other words the number of multiple overlaps at that point. Since it corresponds to the data density at that point, it determines its color in the visualization. For a set of overlapping rectangles forming an overlap O, the *z-index* value z_O represents the number of rectangles in the set, otherwise referred to as the set's *cardinality*. Similarly, for a cell C_i in a grid, we define its *z-index* z_{C_i} as the number of rectangles overlapping with it.

Fig. 2 shows an example of the *z-index*. 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:

$$S_O = width_O \times height_O \quad (6)$$

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

2.2. Problem definition

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

Problem 1. Given a set of rectangles $\mathscr{R} = \{R_1, R_2, ..., R_{n_R}\}$ in an observation space $\Omega = \{(x, y) \in \mathbb{R}^2 \mid 0 \leq x \leq l \text{ and } 0 \leq y \leq l\}$, find the *z-index* $z_{(x,y)}$ of each point $(x, y) \in \Omega$.

It is important to note here that all points of an area that represents a *multiple-overlap* will have the same *z-index*. Since the goal is a density data visualization, we do not need to discretize the space Ω into individual points and calculate all $z_{(x,y)}$ individually. We merely need to produce a set of polygons that, when drawn, cover the entirety of Ω and produce a visual result that matches the corresponding areas of multiple overlaps. Furthermore, using axis-aligned rectangles instead of arbitrary polygons greatly simplifies the drawing process.

In this work, we present two approaches to produce this set of rectangles: (i) a *grid-based* approximate method that relies on partitioning the space Ω and computing multiple overlap areas using the grid cells; (ii) OL-HeatMap, our proposed exact method that uses a sweep-line algorithm and an intersection graph.

3. Methodology

In this section, we present the steps required for the visualization of bounding box heat-maps using the different approaches mentioned in the previous section. 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.

3.1. Grid-based overlap detection

Algorithm 1: Grid-Based.

Input: Set $\mathscr{R}$ of n rectangles, sequence C of g^2 grid cells, observation space size l
Output: Sequence $\mathscr{O}$ of z_{C_i} values corresponding to the grid cells in C

```
cell_size = l/g
for row = 0 to g-1 do      // iterate over cells to create grid
    for col = 0 to g-1 do
        cell.i = row * g + col
        cell.x^0 = col * cell_size;   cell.x^1 = (col + 1) * cell_size
        cell.y^0 = row * cell_size;   cell.y^1 = (row + 1) * cell_size
        C.append(cell)

rect_id = 0
for rect in R do        // insert rectangles into R-tree grid
    RTree.insert(rect_id, rect)
    index += 1

for cell in C do            // query for overlaps at each cell
    O = RTree.queryAt(cell.x^0, cell.x^1, cell.y^0, cell.y^1)
    cell.z = length(O)
    O.append(cell.z)
```

In the uniform grid-based approach, the observation space Ω 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 corresponds to the granularity of the grid.

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*, z_{C_i}), so that we can assign a 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* [6]. 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. The details of the grid-based method are provided in Algorithm 1.

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 every point in the cell (or simply the cell itself). 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. 2 for different values of grid size g. As is apparent in the figure, higher granularity produces visualizations with greater accuracy, i.e., much closer to the original rectangle overlaps. However, as g increases, the number of grid cells (and therefore R-tree queries) increases quadratically. 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 *little* actual meaningful information. Therefore, the total computation cost of the grid-based visualization is $O(n \log n + g^2(\log n + k))$. This often results in situations where, depending on the value of grid size g selected, the end product is either significantly low-accuracy or particularly slow execution times. This trade-off between running time and accuracy is illustrated in the example of Fig. 2.

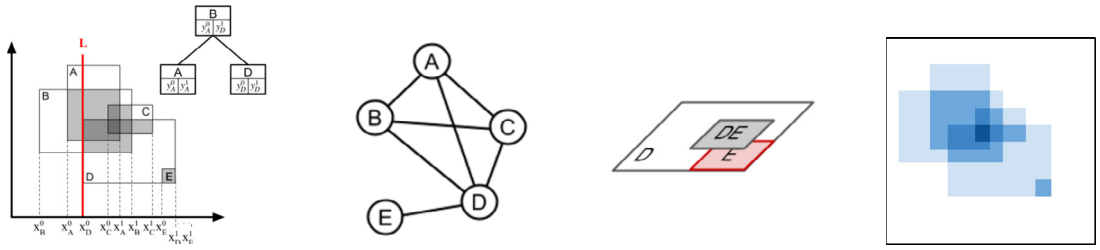
3.2. OL-HeatMap

A different approach to the problem is to take advantage of the fact that the objects to be drawn may overlap and some of them are found in the *foreground* (in other words, are visible) as specified by their *depth* parameter. That allows one to visualize the data by first identifying the *exact location, size* and *z-index* of any overlaps within the data-set. This means that for every set of multiple overlapping rectangles, the details of those overlaps need to be known. Once again, it is possible to address this problem in a brute-force way by finding overlapping pairs of rectangles, then finding overlapping triplets of rectangles, and so on. As is apparent, the computational cost of the brute-force method increases exponentially with the number of rectangles and quickly becomes unfeasible.

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 *plane-sweep* algorithmic paradigm [9]. 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 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 in order. During the sweep, the *Y-dimensional* components of the *active regions* (i.e., the ones that line L is currently traversing over) are maintained in an interval 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 example of the process can be seen in Fig. 3.

The process mentioned above identifies and reports all the pairs of overlapping rectangles in the data-set, and the position of each of those overlaps. However, to address the problem of interest, it is necessary to find not only overlapping pairs, but also the areas where more than two rectangles overlap. A recently proposed variation of the sweep-line algorithm called SLIG (Sweep Line using Intersection Graph) can identify all multiple overlaps in a set of regions (axis-aligned shapes) by utilizing a data structure known as the Region Intersection Graph (RIG) [8]. A Region Intersection Graph (RIG) is a graph where each vertex corresponds to a region in the data-set and a connection between two vertices exists if and only if the respective regions are overlapping. Given a RIG constructed during the sweep using the identified intersection pairs, the problem of identifying multiple overlapping regions now becomes equivalent to that of enumerating all the possible cliques in the RIG graph, a well-studied problem in graph theory with several well-established state-of-the-art algorithms available [10,11]. SLIG is a general algorithm and can be adopted in many domain-specific problems.

We can now present OL-HeatMap, our approach to solving the rectangle heat-map visualization problem. OL-HeatMap is relying on SLIG, but needs to accommodate the specifics of the problem of interest. The overview of the algorithm is as follows (three steps): firstly, it uses a conceptual sweep-line over the observation area



(a) sweep-line, interval tree (b) intersection graph (c) filter covered (d) result visualization

Fig. 3. Overview of the OL-HeatMap method.

Algorithm 2: OL-HeatMap.

Input: Set $\mathscr{R}$ of n rectangles
Output: Set $\mathscr{O} = \{O_1, O_2, \ldots\}$ of visible multiple overlap rectangles, with corresponding *z-index* values z_{O_i}

```
for R in ℛ do        // get start and end points along sweep dimension
    Points.append(x_R^0, x_r^1)
Points.sort()
Graph = SLIG.GetIntersectionGraph(Points)
Cliques = EnumerateVisibleCliques(Graph)
for O in Cliques do   // z-index is clique size
    𝒪.append( (O, z_O) )
```

Procedure EnumerateVisibleCliques(Graph).

```
Visited = {};  Nbrs = {}   // empty dictionaries
for u in Graph do   // Get all neighbors based on order
    Visited.append(u)
    Nbrs[u] = {G.neighbors(u) - Visited}
// Deque: list-like, fast appends, pops on either end
Queue = deque(([u], sorted(nbrs[u])) for u in Graph)
while Queue not empty do
    O_old, Cnbrs_i = queue.popleft()
    visible = True;
    for u in Cnbrs do   // Iterate over common neighbors
        O_new = O_old + [u] // New clique, prepare larger ones
        Queue.append((O_old, O_new, {nbrs[u] if Cnbrs_j ∈ nbrs[u], j > i}))
        if length(O_new) > 1 ∧ S_O_new == S_O_old then   // cover check
            visible = False
    if visible ∧ length(O) > 1 then   // found visible rectangle
        𝒪.append(O_old)
```

Ω. Secondly, it constructs a Region Intersection Graph (RIG) where each vertex represents a rectangle. Finally, it utilizes a variant of a state-of-the-art clique-enumeration algorithm [10,11] to enumerate overlaps to be drawn. In addition, while SLIG identifies and retrieves *all* possible overlaps of multiple rectangles, OL-HeatMap will only need to find the rectangles that are *visible* when drawn as a heat-map. This observation suggests running time savings, since given a set or rectangles $\mathscr{R}$ as input, our final goal is to effectively draw only *visible* rectangles and rectangle overlaps, with a color value corresponding to the number of overlapping rectangles in each area. More formally, if a rectangle R_1 lies within or exactly on another rectangle R_2 (in other words $x^0_{R_2} \leq x^0_{R_1}$, $y^0_{R_2} \leq y^0_{R_1}$ and $x^1_{R_2} \geq x^1_{R_1}$, $y^1_{R_2} \geq y^1_{R_1}$), then their area of overlap $O_{R_1R_2}$ has the exact same position and area as R_1, effectively "covering" it. In the example of Fig. 3, this can be seen with the rectangles E and D. It is useful to note that since:

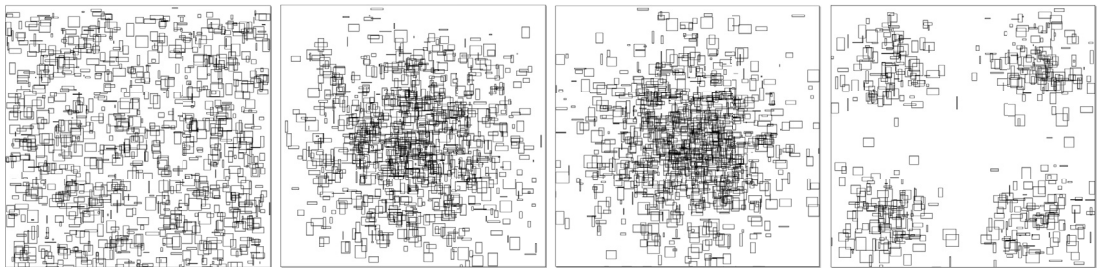
$$S_{O_{R_1R_2}} = width_{O_{R_1R_2}} \cdot height_{O_{R_1R_2}}$$
$$= min(width_{R_1}, width_{R_2}) \cdot min(height_{R_1}, height_{R_2}),$$
$$width_{R_1} \leq width_{R_2}, \quad height_{R_1} \leq height_{R_2}, \quad \text{then}$$
$$S_{O_{R_1R_2}} = width_{R_1} \cdot height_{R_1} = S_{O_{R_1}}$$

In other words, a rectangle is *covered* if and only if it has the same size as its overlap. In these cases, it is not necessary to draw the non-visible rectangle.

OL-HeatMap employs a variant of SLIG, with the same sweep-line step but an altered clique enumeration algorithm. Specifically, while constructing the intersection graph, we maintain information related to the area of each overlap S_O in the corresponding nodes. Afterwards, we calculate the (x, y) coordinates and size S_O of each resulting overlap, along with its z-index value. During the clique enumeration step, when transitioning to a larger clique O_{new}, this allows us to compare the surface areas of the new multiple overlap with that of each smaller clique/rectangle O_i. If any of the rectangles O_i have the same size S_{O_i} as S_{new}, they are marked as *covered* and are filtered out of the output results. Finally, to produce the heat-map visualization, all that's 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 (i.e., are drawn last). An overview of the method's steps and an illustrative visualization example is shown in Fig. 3. The pseudocode of the OL-HeatMap is provided in Algorithm 2.

Computational Complexity: The computational cost of OL-HeatMap is dominated by SLIG and is $O(n \log n + n \cdot c_{max})$, where c_{max} is the number of maximal cliques found in the rectangle intersection graph, or otherwise the number of unique sets of multiple overlapping rectangles that themselves are not all overlapping with other rectangles. The worst-case scenario for this algorithm is the same as the grid-based one, where all rectangles are overlapping at some point. However, this is a degenerate case as discussed earlier.

Advantages: OL-HeatMap offers several advantages to the grid-based approach. First, as it does not involve the use of a grid, its runtime cost is much smaller (the g^2 multiplier of the computation complexity is dropped). In practice, unless the grid size is really small, OL-HeatMap can offer substantial runtime performance gains. Furthermore, OL-HeatMap is an *exact* method, therefore our approach always computes the exact solution and leads to the right visualization. This is in contrast to the grid-based method that only provides an approximate solution, the quality of which depends on the size of the grid utilized. Recall that the grid-based method depicts a tradeoff between accuracy and runtime performance – the larger the grid the more accurate the visualization, but also the more time-consuming. Trying to achieve the exact solution provided of OL-HeatMap, a grid-based approach would need to utilize a theoretical infinite grid, which would also render the runtime cost prohibitive. A comprehensive evaluation is provided in section 4.



(a) uniform (b) triangular (c) gaussian (d) bi-modal

Fig. 4. Synthetically generated data-sets with various distributions (examples).

Table 2
Summary of data-sets. The numbers represent the *actual number of overlaps* found.

Density	$n = 100$			$n = 500$			$n = 1000$			$n = 1500$			$n = 2000$		
Distribution	*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

4. Experimental evaluation

In this section we describe the design and execution of the experimental evaluation for the different methods mentioned. Details on the data-sets and computational environment used are provided, and a comparison of performance and accuracy is presented for the OL-HeatMap and baseline grid-based methods.

4.1. Environment

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

4.2. Data

To evaluate the performance of OL-HeatMap we rely on **both synthetic and real-world data**. The synthetic data allows to examine the behavior of OL-HeatMap (and its competitor methods) under a wide variety of conditions. The real-world data is used to demonstrate the flexibility of OL-HeatMap or how easy it is to adapt it in different domains and for variable definitions of a "rectangle". Real-world data is also used to demonstrate how grid-based methods (that are approximate) can draw a misleading conception about a situation.

For the **synthetic data**, a data generator was implemented that produces data sets with specific characteristics thanks to a controlled number of parameters. For a square space with side l, n rectangles were randomly generated with (x, y) coordinates 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 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 combination 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*, *max length ratio r* and *spatial distribution*. Table 2 provides a summary of the synthetic datasets employed and the number of resulting overlaps for varying values of the these properties. An examples of these distributions are shown on Fig. 4.

For the **real-world scenarios**, we use data from the Storm Event Database [12], an official publication of the National Oceanic and Atmospheric Administration (NOAA). It contains data related to extreme natural events recorded in the United States of America for the last 70 years, with details on tornadoes, storms, hail storms and snowfall. We use this dataset to demonstrate the flexibility of OL-HeatMap, where we are able to represent extreme natural events as rectangles spanning a geographic area. Then we can apply OL-HeatMap (or competitor methods) to extract meaningful insights. Events (rectangles) defined in the Storm Events Database have density comparable to the "dense" synthetic datasets, and follow a roughly uniform distribution with potentially a small number of overlap "hot-spots", similar to the ones in the bi-modal one.

4.3. Accuracy evaluation metrics

The baseline grid-based approach described in Section 3 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. 2, a larger grid size 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 evaluation metrics that quantify the accuracy of grid-based overlap visualizations:

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 has a completely correct z-index value (and therefore color). If a grid cell contains any rectangle boundaries, not all points within the cell should have the same z-

index value; however, as a single grid cell can have a single z-index value, it cannot represent the z-index values (and therefore colors) of its points correctly. A single cell C_i is considered completely correct if, in the original data-set, all the points (x, y) within that cell have the same z-index value as the cell. Formally, the accuracy of that cells can be defined as:

$$AC_{C_i} = \begin{cases} 1 \text{ if } z_{C_i} = z_{(x,y)}, \forall\, (x, y) \in C_i \\ 0 \text{ otherwise} \end{cases}$$

With overall grid accuracy being:

$$AC_{grid} = \frac{\sum_{i=1}^{g^2} AC_{C_i}}{g^2} \cdot 100\%$$

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). The definition of this area-based accuracy metric for a single cell is:

$$AS_{C_i} = \frac{\sum_{\forall R_k} S_{C_i,R_k}}{|C|}$$

where R_k are all rectangles with the same z-index value as C_i and S_{C_i,R_k} is the overlap of these rectangles with the grid cell. Overall grid accuracy is once again:

$$AS_{grid} = \frac{\sum_{i=1}^{g^2} AS_{C_i}}{g^2} \cdot 100\%$$

As the second metric is more refined, we only use that one for reporting the accuracy of a grid-based visualization in the experiments.

4.4. Experiments

Our experiments aim to evaluate the following aspects:

- **OL-HeatMap Accuracy Performance (versus Grid-Based Methods)** While OL-HeatMap is an exact method, the accuracy of grid-based methods depends on the grid size. What is the effect of the grid size in the accuracy of the grid-based method, for data-sets of varying 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 varying size and distribution?
- **OL-HeatMap Scalability** How does our proposed method OL-HeatMap scale for data-sets of larger sizes, compared to the grid-based method?
- **OL-HeatMap Flexibility** OL-HeatMap can be applied for the visualization of various real-world datasets from various domains, as explained in Section 1. To demonstrate its versatility, we apply OL-HeatMap to visualize multiple real-world datasets.

4.4.1. OL-HeatMap *accuracy performance (versus grid-based methods)*

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 $n = 1000$ rectangles. Furthermore, we measured the resulting accuracy for data-sets of different sizes, this time with the same density "dense" and once again uniformly distributed. The results for these experiments can be seen in Fig. 5a and Fig. 5b, respectively.

As expected, larger grids 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 matched only by very large grids.

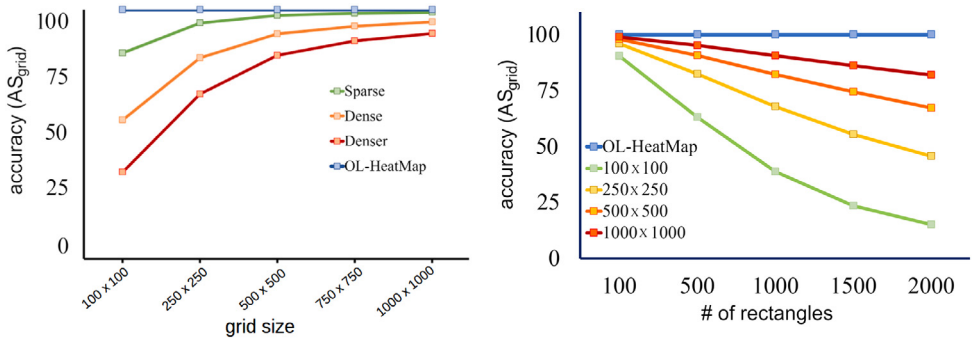
4.4.2. OL-HeatMap *runtime performance*

We evaluated the time performance of OL-HeatMap against the baseline grid-based method, as a function of the number of rectangles n 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 number of rectangles n is fixed to 1000 in the second, while in both cases the data-sets are of "dense" density.

As can be seen in Fig. 6a and Fig. 6b, in most cases OL-HeatMap outperforms the baseline for all but the smallest grid sizes, which were the ones with the lowest accuracy. This highlights the trade-off between time and accuracy in the grid-based algorithm, and shows that OL-Heatmap should outperform it in either or both metrics. 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, which has accuracy comparable to OL-HeatMap. Furthermore, as discussed earlier, real-world data have distributions closer to the uniform or bi-modal ones with many "hot-spots", whereas concentrated gaussian distributions with most objects overlapping together are less meaningful.

4.4.3. OL-HeatMap *scalability*

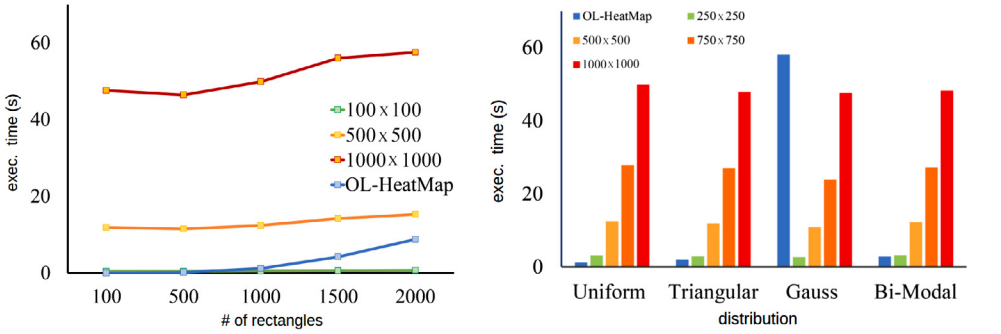
So far, we have limited our comparative performance analysis to smaller data-sets, up-to 2000 overlapping rectangles, in order to ensure quicker data visualization. Here, we examine the scalability of both algorithms for much larger number of overlapping objects. As mentioned before, there is a trade-off between speed and accuracy in the grid-based algorithm. Therefore, we select appropriate grid sizes to produce two baselines: grids producing 75% and 95% area accuracy AS_{grid}. We compared the execution time of OL-HeatMap against those two baselines for 100 up to 10^5 rectangles using a *dense* dataset following the *bi-modal* distribution. Fig. 7



(a) Different grid sizes and densities for $n = 1000$, in a uniform distribution

(b) Different grid sizes and numbers of objects, in a *dense* uniform distribution

Fig. 5. Accuracy of grid-based method vs OL-HeatMap's exact results. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)



(a) Different grid sizes and numbers of objects, in a *dense* uniform distribution

(b) Different grid sizes and]em dense distributions for $n = 1000$

Fig. 6. OL-HeatMap vs. grid-based time performance comparison.

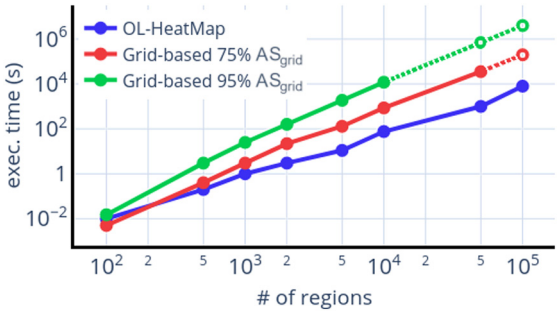


Fig. 7. OL-HeatMap vs. grid-based times for 10^2 to 10^5 objects in *dense* bi-modal distribution.

shows the results in logarithmic scale. OL-HeatMap consistently outperforms the 75%- and 95%- accuracy grid-based approaches by **one and two orders of magnitude**, respectively. This is a significant improvement, allowing for the visualization of very large numbers of objects in reasonable execution time and while always being 100% accurate. On the other hand, for the case of 10^5 rectangles, the less accurate grid-based method (75% accurate) would require more than 10^5 seconds ($\approx$ 28 hours) to complete (best estimates). The execution time would be even longer if we require an accuracy of 95%.

4.4.4. OL-HeatMap *flexibility*

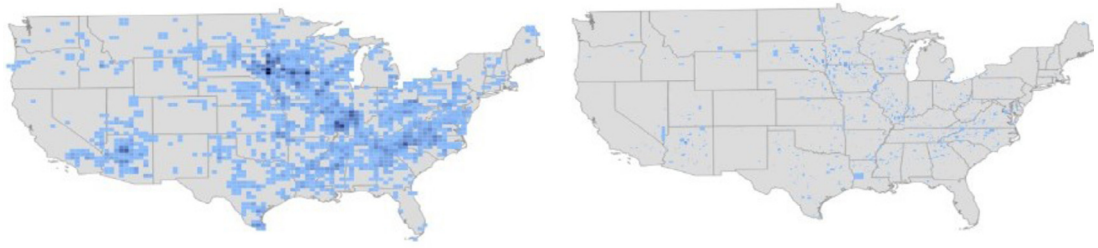
We present **two cases** that utilize different information in the Storm Event data-set to highlight the strengths of OL-HeatMap. In both cases, we represent each event as a rectangle defined by an event's **beginning** and **end coordinates**.

Table 3
Overview of overlaps and accuracy of storm data.

	US (2017-2018)	Florida (1953-2018)
Total overlaps	130651	43834
Visible overlaps	94621	32774
Accuracy (AC_{grid})	96.26%	71.37%
Accuracy (AS_{grid})	96.35%	75.78%

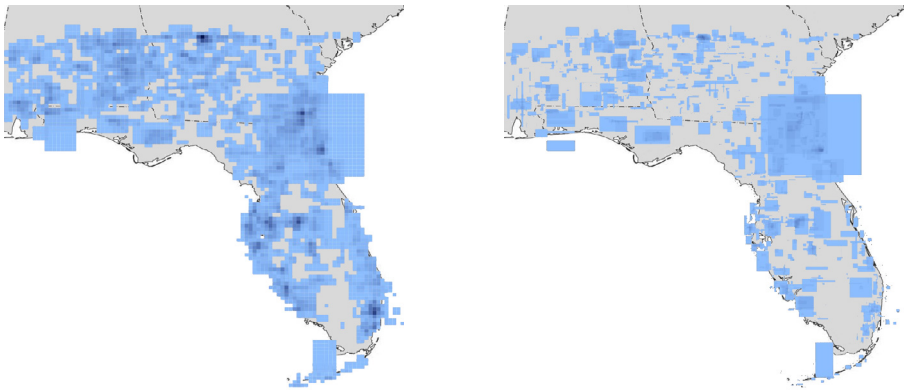
(a) Storms in US from 2017 to 2018. For this analysis, we examine all Tornado events in the United States for the years 2017 and 2018. After constructing the rectangles, we use both the *grid-based* and OL-HeatMap methods to find the density of overlaps and visualize them. In the resulting visualization, denser zones indicate a higher amount of storms taking place in different areas within the US. Fig. 8 showcases the resulting heat-maps produced. The interesting trend to note from these visualizations is 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 due to the fact that the highest concentration of Tornadoes in the US happens here. Comparing the visualizations produced by OL-HeatMap versus the less accurate grid-based method reveals a more accurate rhetoric about the area – only smaller, very specific areas hold to the characterization. Table 3 shows the accuracy results of the grid-based method.

(b) Hurricanes in Florida from 1953 to 2018. We provide an overview of all the hurricanes happening in Florida from 1953 to 2018. We form individual rectangles for each recorded event by using the **beginning and end coordinates** and used OL-HeatMap to effectively find and visualize the events. Every year Florida is ravaged by catastrophic natural calamities. An estimated 123 billion dollars worth of damage have been recorded from 2000 till to this



(a) Grid-based 100×100 (b) OL-HeatMap

Fig. 8. Visualization of all storms in US from 2017-18.



(a) Grid-based 100×100 (b) OL-HeatMap

Fig. 9. Visualization of all hurricane events in Florida from 1953-2018.

date. As United States' densest hurricane zone, we visualized each hurricane as its own rectangle, found overlaps and subsequently visualized using OL-HeatMap. The data contains all recorded hurricanes from 1953 to 2018 and each rectangle represent the area of impact for each hurricane. Table 3 shows the accuracy results of the grid-based method.

As can be seen from Fig. 9, we determine the locations of our bounding boxes to be south of 82 Degrees Latitude, which would be north of the Florida State. All Hurricane events which has a begin-latitude value of 82 have been counted in as an event hurling towards or originated in Florida. From Fig. 9, it can be noted 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. Fig. 9a shows the grid based visualization of Florida and Fig. 9b shows the OL-HeatMap of the storms in Florida since 1953. As can be seen from Fig. 9a, a grid-based visualization of all the hurricane events provides a view of a hurricane-ravaged Florida, where approximately the entire state (and its neighbors) have been hit. However, a more accurate visualization provided by OL-HeatMap paints a completely different picture, where it can actually be seen that Florida has several much more limited areas where hurricanes have hit the hardest.

5. Proof-of-concept demo system

In this section, we discuss the demo dashboard 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 them accordingly.

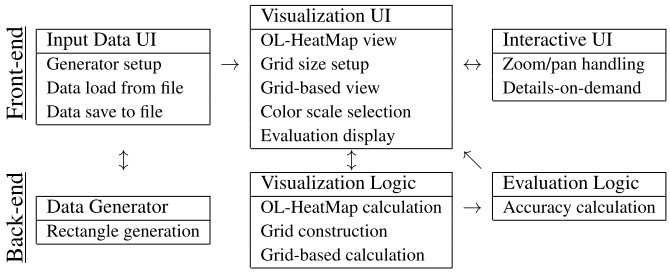


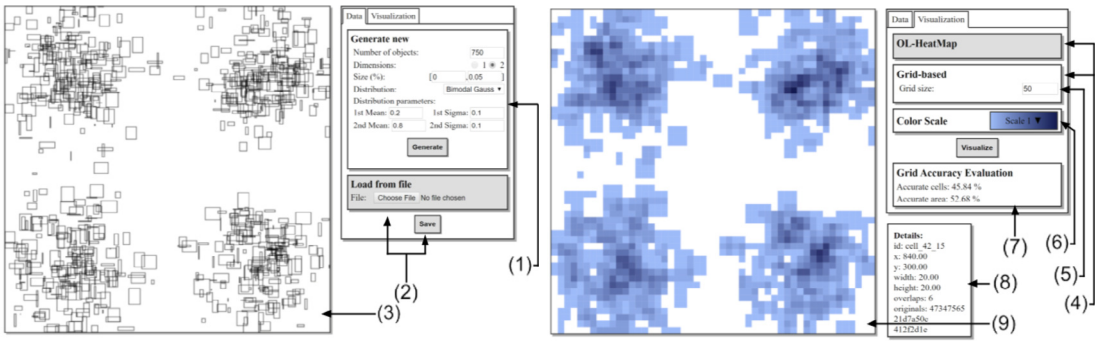
Fig. 10. System architecture overview.

5.1. System architecture overview

We provide an overview 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. Fig. 10 provides an overview of the architecture.

5.1.1. 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. The interface allows the user to generate and store synthetic data-sets by specifying parameter values for the random data generator through 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 size value g, as well as the color scale used for



(a) Data input (b) Grid-based visualization

Fig. 11. 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.

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, including both AC_{grid} and AS_{grid}. The user interface views for the data loading/generation and visualization can be seen in Fig. 11.

5.1.2. Back-end

The dashboard demo is structured as a lightweight WebApp-style application, with a Python Flask 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 visualization are calculated using OL-HeatMap. Finally, the accuracy evaluation scores for the grid-based approach are shown; recall that the OL-HeatMap method by definition produces exact results.

5.2. Visualization limitations

The visualization of OL-HeatMap inherits limitations of browser-based systems, including limitations of the drawing methodology and data loading. The main limitation is due to the limited ability of the browser to utilize all available GPU resources. D3 uses HTML5 GPU acceleration, but better performance could potentially be achieved with a more low-level interaction with the GPU. Our work is orthogonal to any (implementation) optimization related to rendering data to graphics and can therefore take advantage of it.

6. Related work

The work in this paper is related to *density-based visualization methods* and methods for computing *rectangle overlaps*. Several key ideas have already been referenced throughout this paper, and here we present a more comprehensive view of existing work on these topics.

6.1. Density-based visualization methods

A density-based visualization is adequate for noticing changes in the data, visualizing clusters and pointing out outliers [13]. There are several proposals which address the topic of representing density. Local data density can be visualized by either aggregation, using area or usage of color [14]. Due to their utility, several methods have been proposed over the years for density-based representations. As standard *scatterplots* started to become obsolete due to big data and overplotting issues [15], de-cluttering methods started being proposed, including adding opacity [16], color, smoothing [13] and/or binning the data. Ellis and Dix [17] discussed these strategies to eliminate visual cluttering. Bertini, Di Girolamo and Santucci discussed about optimizing the visualization process by using various quality metrics and density distribution [18].

Kernel Density Estimation (KDE), along with its variants, such as Approximate KDE (AKDE) [19] 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 dimensions and there are constraints on sub-bandwidth [21].

Sampling-Based Approximation Methods: 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. [22] discussed 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. Local density can also be measured by stacking of visual components of overlapping cases, as discussed by Dang, Wilkinson and Anand [14]. Van Liere and De Leeuw [23] use graphs to compute density in irregularly-sampled data. They find the underlying density by transforming a graph into two-dimensional scalar fields and rendering the graph into a color-coded map.

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 [24,25]. An *antichain* can be selected which serves as an abstraction. This abstraction can be used to make the visualization interactive as well. Cottam, Lumsdaine and Wang provide an unified solution by creating visual abstractions to avoid overplotting [26].

Binned Aggregation: The cost of KDE can be avoided by using binned aggregation. Liu, Jiang and Heer proposed *ImMens* [27] which groups data points into predefined "bins". These "bins" are not dependent on others, 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 with KDE-based methodologies has been used to visualize dense time series [13] [20].

Distributed Methods for Density Visualization: Big data architectures are used to scale up to larger data-sets. For large spatio-temporal data-sets, a specialized system [28] has been proposed to process data and render images of heat-maps. The MapReduce framework has also been used [29] to distribute existing algorithms. Perrot, Bourqui, Hanusse and Auber discussed using the Apache Spark framework to perform canopy clustering in order to create low-latency heat-maps [30]. For large temporal data-sets, Spectogram has also been proposed as a tool to visualize high density data [31].

6.2. 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. We discuss a few methods that have been proposed in this area.

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 $O(n^2)$ [7]. 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. In this case, sweep-line is used as a part of the process of detecting intersections. A tree-based data structure, such as an interval tree, can be used simultaneously for better performance [32]. However, for denser data-sets, this method is computationally expensive. The resulting structure of the intersections and overlays is defined as "arrangement". Alt and Schraf count *arrangements* in AABBs by designing parallel sweep-line-based methods [33].

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 cell. The idea of a uniform grid has been proposed to detect collisions [4], 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, Rak and Calvin proposed a 2^d data structure for dynamic adjusting of the grid cell size [34]. Antochi, Juurlink, Vassiliadis and Liuha discussed about optimizing tile based rendering to improve overlap detection [35]. One of the interesting applications of uniform grid can be found in geotechnical engineering[36], where a grid-based approach outperforms Binary Volume Hierarchy based collision detection from a memory perspective.

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 [6]. R-tree helps to group the objects into bounding rectangles of increasing size. Other methods include a *range tree* based algorithms [37], or streaming algorithms [38] which requires huge memory space to build and store the range tree.

7. Conclusion

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, including both synthetic and real-world data-sets. 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. We can approximate different shapes with rectangles as well. 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.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] T. Matsuyama, M. Nagao, et al., A file organization for geographic information systems based on spatial proximity, Comput. Vis. Graph. Image Process. 26 (3) (1984) 303–318.
[2] D. Papadias, Y. Theodoridis, Spatial relations, minimum bounding rectangles, and spatial data structures, Int. J. Geogr. Inf. Sci. 11 (2) (1997) 111–138.
[3] J. Fang, J. Wong, K. Zhang, P. Tang, A new fast constraint graph generation algorithm for VLSI layout compaction, in: IEEE International Symposium on Circuits and Systems, 1991.
[4] W.R. Franklin, C. Narayanaswaml, M. Kankanhalll, D. Sun, M. chu Zhou, P.Y. Wu, Uniform grids: a technique for intersection detection on serial and parallel machines, in: Proceedings of Auto-Carto 9, Springer-Verlag, Baltimore, Maryland, US, 1989, p. 100109.
[5] Y. Nekrich, A linear space data structure for orthogonal range reporting and emptiness queries, Int. J. Comput. Geom. Appl. 19 (01) (2009) 1–15, https://doi.org/10.1142/s0218195909002800.
[6] A. Guttman, R-trees: a dynamic index structure for spatial searching, SIGMOD Rec. 14 (2) (1984) 47–57, https://doi.org/10.1145/971697.602266, http://doi.acm.org/10.1145/971697.602266.
[7] M.I. Shamos, D. Hoey, Geometric intersection problems, in: 17th Annual Symposium on Foundations of Computer Science, IEEE, Houston, Texas, 1976, pp. 208–215.
[8] T. Pechlivanoglou, V. Chu, M. 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, 2019.
[9] J.L. Bentley, T.A. Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Trans. Comput. C-28 (9) (1979) 643–647.
[10] D. Eppstein, M. Löffler, D. Strash, Listing all maximal cliques in sparse graphs in near-optimal time, in: Algorithms and Computation, in: Lecture Notes in Computer Science, vol. abs/1006, 2010, pp. 403–414.

[11] C. Bron, J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, Commun. ACM 16 (9) (1973) 575–577.
[12] N. C. for Environmental Information, Storm events database, https://www.ncdc.noaa.gov, 1953.
[13] H. Wickham, Bin-summarise-smooth: a framework for visualising large data, Tech. rep, University of Auckland, 2013.
[14] T.N. Dang, L. Wilkinson, A. Anand, Stacking graphic elements to avoid overplotting, IEEE Trans. Vis. Comput. Graph. 16 (6) (2010) 1044–1052.
[15] D.B. Carr, R.J. Littlefield, W.L. Nicholson, J.S. Littlefield, Scatterplot matrix techniques for large n, J. Am. Stat. Assoc. 82 (398) (1987) 424–436.
[16] J. Matejka, F. Anderson, G. Fitzmaurice, Dynamic opacity optimization for scatter plots, in: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, ACM, 2015, pp. 2707–2710.
[17] G. Ellis, A. Dix, A taxonomy of clutter reduction for information visualisation, IEEE Trans. Vis. Comput. Graph. 13 (6) (2007) 1216.
[18] E. Bertini, A. Di Girolamo, G. Santucci, See what you know: analyzing data distribution to improve density map visualization, in: EuroVis, vol. 7, 2007, pp. 163–170.
[19] A. Perrot, R. Bourqui, N. Hanusse, F. Lalanne, D. Auber, Large interactive visualization of density functions on big data infrastructure, in: 2015 IEEE 5th Symposium on Large Data Analysis and Visualization, LDAV, IEEE, Chicago, Illinois, USA, 2015, pp. 99–106.
[20] O.D. Lampe, H. Hauser, Curve density estimates, in: Computer Graphics Forum, in: Wiley Online Library, vol. 30, The Eurographics Association and Blackwell Publishing Ltd, Llandudno, UK, 2011, pp. 633–642.
[21] C. Li, G. Baciu, Y. Han, Interactive visualization of high density streaming points with heat-map, in: 2014 International Conference on Smart Computing, IEEE, Hong Kong, China, 2014, pp. 145–149.
[22] H. Chen, W. Chen, H. Mei, Z. Liu, K. Zhou, W. Chen, W. Gu, K. Ma, Visual abstraction and exploration of multi-class scatterplots, IEEE Trans. Vis. Comput. Graph. 20 (12) (2014) 83–92.
[23] R. Van Liere, W. De Leeuw Graphsplatting, Visualizing graphs as continuous fields, IEEE Trans. Vis. Comput. Graph. 9 (2) (2003) 206–212.
[24] D. Auber, F. Jourdan, Interactive refinement of multi-scale network clusterings, in: Ninth International Conference on Information Visualisation, IV'05, IEEE, London, UK, 2005, pp. 703–709.
[25] J.-Y. Delort, Vizualizing large spatial datasets in interactive maps, in: 2010 Second International Conference on Advanced Geographic Information Systems, Applications, and Services, IEEE, IEEE, St. Maarten, Netherlands Antilles, 2010, pp. 33–38.
[26] J. Cottam, A. Lumsdaine, P. Wang, Overplotting: unified solutions under abstract rendering, in: 2013 IEEE International Conference on Big Data, IEEE, 2013, pp. 9–16.
[27] Z. Liu, B. Jiang, J. Heer, Immens: real-time visual querying of big data, in: Computer Graphics Forum, in: Wiley Online Library, The Eurographics Association and, vol. 32, John Wiley Sons Ltd, Leipzig, Germany, 2013, pp. 421–430.
[28] A. Eldawy, M.F. Mokbel, S. Alharthi, A. Alzaidy, K. Tarek, S. Ghani, Shahed: a MapReduce-based system for querying and visualizing spatio-temporal satellite data, in: 2015 IEEE 31st International Conference on Data Engineering, IEEE, IEEE, Seoul, South Korea, 2015, pp. 1585–1596.
[29] H.T. Vo, J. Bronson, B. Summa, J.L. Comba, J. Freire, B. Howe, V. Pascucci, C.T. Silva, Parallel visualization on large clusters using MapReduce, in: 2011 IEEE Symposium on Large Data Analysis and Visualization, IEEE, IEEE, Providence, Rhode Island, USA, 2011, pp. 81–88.
[30] A. Perrot, R. Bourqui, N. Hanusse, D. Auber, Heatpipe: high throughput, low latency big data heatmap with spark streaming, in: 2017 21st International Conference Information Visualisation, IV, IEEE, London,UK, 2017, pp. 66–71.
[31] S. Zarina, O. Krasts, Spectrogram based toolkit for high density visualization of data, in: 2016 International Conference on Computational Science and Computational Intelligence, CSCI, 2016, pp. 1393–1394.
[32] J.D. Cohen, M.C. Lin, D. Manocha, M. Ponamgi, 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, ACM, New York, NY, USA, 1995, pp. 189–ff.
[33] H. Alt, L. Scharf, Computing the depth of an arrangement of axis-aligned rectangles in parallel, in: Abstracts 26th European Workshop on Computational Geometry, 2010, pp. 33–36.
[34] D.J. Van Hook, S.J. Rak, J.O. Calvin, Approaches to RTI implementation of HLA data distribution management services, in: Proceedings of the 15th DIS Workshop, 1996, pp. 535–544.
[35] I. Antochi, B. Juurlink, S. Vassiliadis, P. Liuha, Efficient tile-aware bounding-box overlap test for tile-based rendering, in: 2004 International Symposium on System-on-Chip, 2004, Proceedings, 2004, pp. 165–168.
[36] R. Lubbe, W.-J. Xu, D.N. Wilke, P. Pizette, N. Govender, Analysis of parallel spatial partitioning algorithms for GPU based DEM, Comput. Geotech. 125 (2020) 103708.
[37] J. Bentley, D. Wood, An optimal worst case algorithm for reporting intersections of rectangles, IEEE Trans. Comput. 29 (07) (1980) 571–577.
[38] A. Zomorodian, H. Edelsbrunner, Fast software for box intersections, Int. J. Comput. Geom. Appl. 12 (1) (2002).