




# SimpleSets: Capturing Categorical Point Patterns with Simple Shapes

Steven van den Broek , Wouter Meulemans , and Bettina Speckmann 

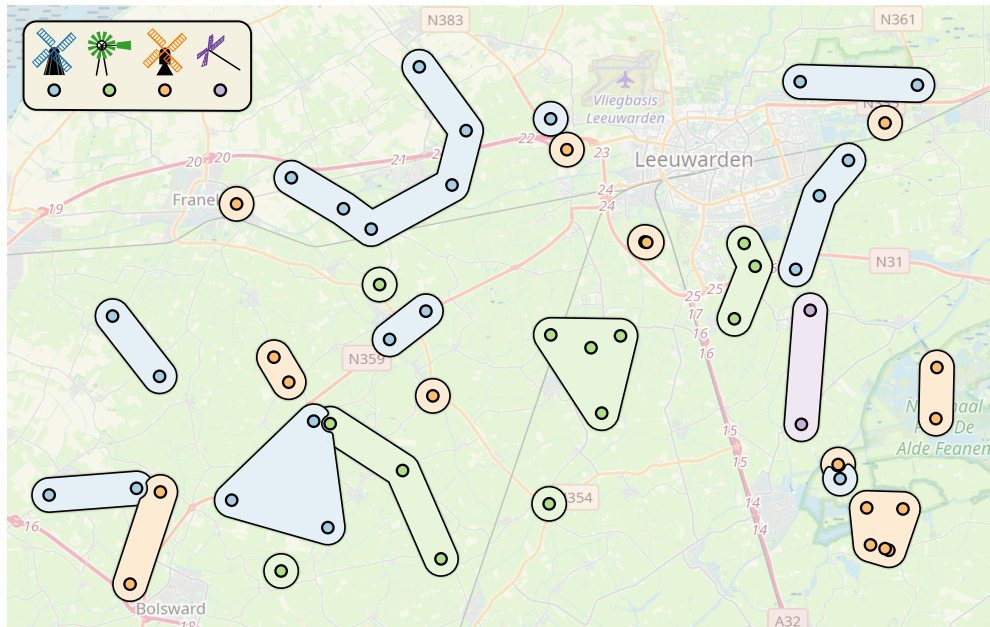


Fig. 1: A SimpleSets visualization of mills around Leeuwarden, The Netherlands. The mill types are: angular mill (blue); vertical wind engine (green); spider head mill (orange); and tjasker (purple). Data by [molendatabase.nl](http://molendatabase.nl) with permission, map from [openstreetmap.org](http://openstreetmap.org).

**Abstract**—Points of interest on a map such as restaurants, hotels, or subway stations, give rise to categorical point data: data that have a fixed location and one or more categorical attributes. Consequently, recent years have seen various set visualization approaches that visually connect points of the same category to support users in understanding the spatial distribution of categories. Existing methods use complex and often highly irregular shapes to connect points of the same category, leading to high cognitive load for the user. In this paper we introduce SimpleSets, which uses simple shapes to enclose categorical point patterns, thereby providing a clean overview of the data distribution. SimpleSets is designed to visualize sets of points with a single categorical attribute; as a result, the point patterns enclosed by SimpleSets form a partition of the data. We give formal definitions of point patterns that correspond to simple shapes and describe an algorithm that partitions categorical points into few such patterns. Our second contribution is a rendering algorithm that transforms a given partition into a clean set of shapes resulting in an aesthetically pleasing set visualization. Our algorithm pays particular attention to resolving intersections between nearby shapes in a consistent manner. We compare SimpleSets to the state-of-the-art set visualizations using standard datasets from the literature.

**Index Terms**—Set visualization, geographic visualization, algorithms

## 1 INTRODUCTION

Categorical point data are a common data type that consist of a location in the plane (or possibly in a higher dimension) labelled with a set of one or more categorical attributes. This data type captures, for example, points of interest on a map, such as restaurants or hotels, or the nodes of an embedded graph such as a social network. In recent years, various approaches have been developed that visually connect the points that belong to the same category. Such connecting shapes aid the user in identifying patterns and understanding the data distribution.

However, the majority of existing methods use one single shape to connect all points in the same category. Since these points can be scattered throughout the plane, the shapes that represent the categories are often complex and may intersect even when the categories have no shared points, leading to a high cognitive load for the user.

We present SimpleSets, which uses simple enclosing shapes to capture patterns in point data with a single categorical attribute (Figure 1). SimpleSets provides a clean overview of the data distribution, highlights natural spatial patterns, and covers little space in addition to the points. The patterns constructed by SimpleSets depend on one intuitive parameter which allows the user to group points that are further apart or closer together in patterns that cover more or less space, as desired.

**Organization.** We begin in Section 2 with an extensive overview of related work. SimpleSets consists of two phases: first, the point data is partitioned into spatial patterns, and then these patterns are drawn as enclosing shapes. In Section 3 we first discuss the overarching design

TU Eindhoven, the Netherlands. E-mail:  
[s.v.d.broek, w.meulemans, b.speckmann]@tue.nl.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication  
xx xxx. 201x; date of current version xx xxx. 201x. For information on  
obtaining reprints of this article, please send e-mail to: reprints@ieee.org.  
Digital Object Identifier: xx.xxx/TVCG.201x.xxxxxx

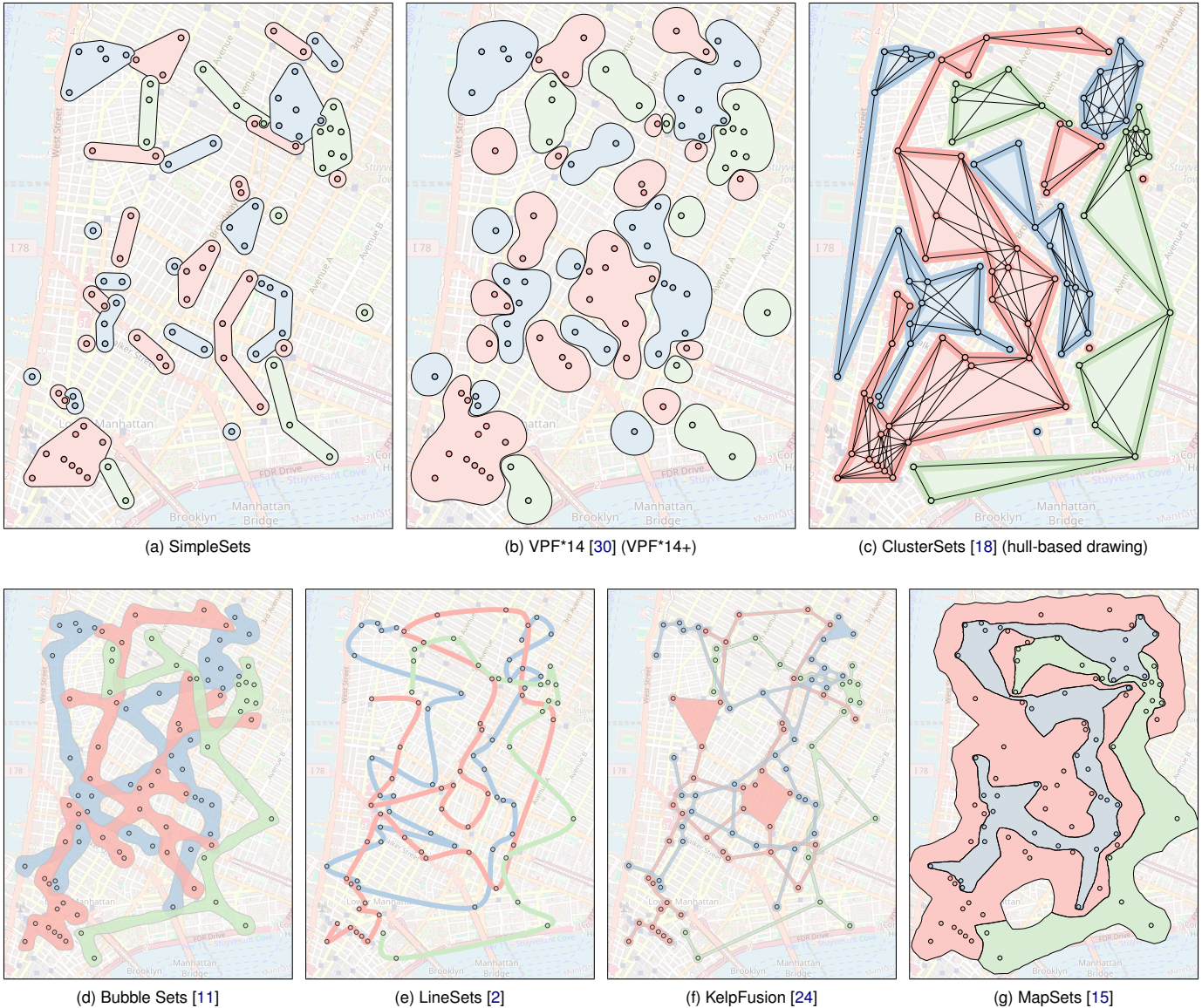


Fig. 2: A comparison of visualizations on a common benchmark dataset originating from the paper that introduced Bubble Sets [11]. The points show hotels (blue), subway entrances (red), and medical clinics (green) in lower Manhattan.

decisions that underlie SimpleSets. Then, in Section 4 we describe how to partition the input into two types of simple patterns: islands and banks. Intuitively, islands are convex clusters of points and banks are sequences of points that do not bend too much. In Section 5, we describe our drawing algorithm in detail; note that we can render any set of spatial point patterns, not only the ones defined by SimpleSets. In the supplementary material we show how to render other patterns from the state-of-the-art with our pipeline. Finally, in Section 6 we discuss SimpleSets visualizations for several datasets from the literature and compare to the state-of-the-art. We implemented our algorithm; our code is open-source to facilitate re-use and reproducibility and is available at [github.com/tue-alga/SimpleSets](https://github.com/tue-alga/SimpleSets) and [doi:10.5281/zenodo.12784670](https://doi.org/10.5281/zenodo.12784670). Our implementation also includes code for the two methods most related to ours: VPF\*14 [30] and ClusterSets [18]; we re-implemented both methods since the original code was not or only partially available. We close with a discussion of possible extensions and future work.

## 2 RELATED WORK

The problem we address in this paper is a set visualization problem with points at predefined positions as elements and categorical attributes

as sets. Set visualization in general is a well-studied problem, and a range of visualization techniques have been used to visualize sets, from the well-known Euler and Venn diagrams, to matrix-based techniques. We refer to the survey by Alsallakh et al. [3] for an overview of set visualization research before 2016, and to Paetzold et al. [25] for more recent results. Convex shapes often feature in set visualizations such as Euler diagrams [21, 25] or node-link diagram overlays [20, 27], as Gestalt theory indicates that they provide a good sense of grouping [31]. This motivates the use of convex islands in our visualization.

In the remainder we discuss work on set visualizations that use predefined positions. Dinkla et al. [13] provide a detailed analysis of this problem. They list various tasks that a user may want to perform using the set visualization, constraints that any visualization should satisfy, and criteria that make shapes effective for set visualization. The most important criteria are:

- C1 Low cognitive load.
- C2 Strong continuation of shapes that depict the same set.
- C3 Little obfuscation of a possible underlying visualization.
- C4 Little distortion of point position and density.

These criteria are illustrated in Figure 3. Most existing approaches

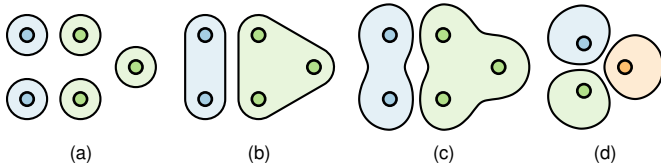


Fig. 3: Figure (c) uses arguably more complex shapes than (b) resulting in higher cognitive load (C1). Figures (b) and (c), compared to (a), use fewer and larger shapes resulting in better continuation (C2) but more obfuscation (C3) and distortion (C4). Figure (d) distorts point position compared to (a); the expected point position is at the centroid of a shape.

focus on the second criterion and use only a single connecting shape which tends to negatively impact the other criteria. SimpleSets instead relies on color to visually connect disjoint shapes that depict the same set, and explicitly connects points only if they form a simple spatial pattern. This allows SimpleSets to do well on the other criteria.

In their survey, Alsallakh et al. [3] identify two types of visual representations of sets: line-based and region-based overlays. We refine the region-based category by distinguishing hull-based and Voronoi-based visualizations. The approaches differ in the size of areas that are colored. Line-based approaches color only a small area around points and around links that connect points. Hull-based approaches color larger regions, and Voronoi-based approaches color the majority of the area being visualized. With the three categories as a guide, we give an overview of related work. Figure 2 shows most visualizations we discuss in the following on a commonly used benchmark dataset; we also include SimpleSets. In Section 6 we discuss the visualizations most similar to SimpleSets in more detail and in comparison to SimpleSets.

**Hull-based.** Byelas and Telea [8] use a convex hull as the basis for a hull-based visualization. For each set, the convex hull of its elements is deformed to create a concave shape enclosing only the points belonging to the set. Collins et al. [11] create similar concave hulls in their Bubble Sets visualization, but in a different way. They construct spanning trees for the sets of points and use them as a basis for a potential field. Isocontours of the potential field form the bubble-like shapes of the Bubble Sets visualization (Figure 2d). Vihrovs et al. [30] and the authors of F2-Bubbles [32] improve upon the potential field defined by Collins et al. In particular, the potential field of Vihrovs et al. guarantees that regions of sets with no common elements do not overlap. Figure 2b shows a visualization of Vihrovs et al. where the potential field is constructed only from the sets of points, not from the spanning trees, resulting in multiple shapes per set.

**Line-based.** Line-based approaches are more minimal than hull-based approaches and align with the minimal ink principle (Tufte’s rule [29]). LineSets [2] represents each set by one thick colored curve (Figure 2e). Kelp Diagrams [13] uses a spanning graph and connect points in a more controlled manner than LineSets. When points form a cluster, line-based approaches are unsatisfactory as one would like to enclose such a cluster with a shape to provide a good sense of grouping [31]. KelpFusion [24] (Figure 2f) is a hybrid between a line-based and hull-based approach. It is similar to Kelp Diagrams, with the main visual difference being that faces of the spanning graph may be filled.

**Voronoi-based.** Line-based visualizations are particularly useful when there is an underlying visualization (base map) that one does not want to obscure. If one is not concerned with this, and points belong to only one category, a simple visualization can be created from a Voronoi diagram [6] of the data points by merging cells of points that belong to the same category. The authors of GMap [17] take this approach. They create a Voronoi diagram of an augmented set of points to construct smooth regions. MapSets [15] (Figure 2g) is similar to GMap, but the authors ensure that the regions representing each category are connected by constructing pairwise non-overlapping spanning trees for each set, and compute a Voronoi diagram based on those trees.

Geiger et al. [18] recently introduced ClusterSets, which can be drawn in all three styles: line-based, hull-based and Voronoi-based. The visualization is based on a proximity graph; Geiger et al. suggest the use of  $\beta$ -skeletons. This proximity graph connects only points of the same category. Geiger et al. compute a planar spanning forest of the proximity graph, minimizing the number of connected components. The planar spanning forest can be drawn directly, yielding a line-based visualization. However, the authors suggest adding back edges of the proximity graph and drawing the boundary polygons as in Figure 2c. They also propose a Voronoi-based drawing where the Voronoi diagram of the polygons is computed and drawn in appropriate colors.

**Hypergraph drawing.** A set visualization can be viewed as a drawing of a hypergraph whose hyperedges are the sets. Several papers study the problem of drawing a spatial hypergraph, a hypergraph whose vertices have a fixed position, as a colored spanning graph. Specifically, they investigate ink minimization of such graphs, both in the setting where crossings are allowed [1, 22] and the setting where the graph is required to be planar [10]. These results are particularly relevant to techniques like Bubble Sets, LineSets, Kelp Diagrams and KelpFusion which use such structures.

**Multiple vs single shape.** The visualizations depicted in the bottom row of Figure 2 all use one shape per category. Hence they generally satisfy C2 well, although to different degrees: LineSets provide less continuation than KelpFusion [13]. However, by using a single shape, the four visualizations inevitably cover parts of the map that are far away from points, and thereby also distort point density (C4). Moreover, the left three visualizations (Bubble Sets, LineSets, and KelpFusion) create many intersections between shapes, which clutter the visualizations, increasing the cognitive load (C1). MapSets create no intersections but are complex and highly irregular. In addition, they heavily obfuscate the base map (C3) and distort point density (C4). The visualizations in the top row of Figure 2 use multiple shapes, which allows them to score well on criteria C1, C3 and C4. In Section 6 we compare these visualizations on more datasets and backed by quantitative measures.

### 3 SIMPLESETS

In this section we describe the overarching design decisions that underlie the two main phases of SimpleSets: constructing a partition into point patterns and drawing such a partition. Figure 4 illustrates our pipeline; the first step of the drawing phase dilates the patterns, this step is explicitly included in the illustration for ease of understanding. We present the algorithmic details for the two phases in Sections 4 and 5.

A *pattern* is a connected region in the plane that contains on its boundary or in its interior a set of data points; all data points in a

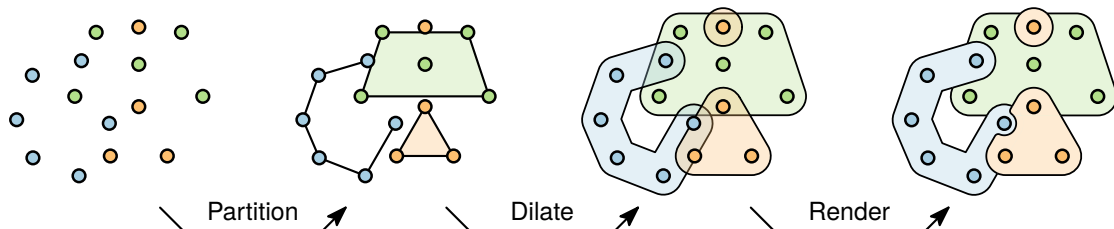


Fig. 4: SimpleSets pipeline. From left to right: data points; partition into patterns; dilated patterns; final enclosing shapes.

pattern share a category. Patterns can be 0-, 1-, or 2-dimensional and may consist of a single point only, polylines, spanning trees, spanning graphs, or polygonal regions. In addition to lines, patterns may be bounded by curves. Two patterns are *disjoint* if their sets of data points are disjoint. Two patterns *overlap* if their connected regions overlap. Note that two disjoint patterns may overlap.

SimpleSets receives categorical point data as input and partitions them into disjoint patterns that do not overlap. Our drawing algorithm, however, can handle both overlapping and non-overlapping disjoint patterns. To illustrate this fact we applied our drawing algorithm to the disjoint patterns that underlie LineSets (overlapping polylines), Bubble Sets (overlapping spanning trees), and ClusterSets (non-overlapping polygons). Figure 6 of the supplementary material shows the results.

### 3.1 Islands and Banks

SimpleSets partitions data points into two types of patterns that correspond to simple shapes: islands and banks. In the following we assume, without loss of generality, that no three data points are collinear.

An *island* [5] is the convex hull of a set of data points. Islands have previously been studied by Bautista-Santiago et al. [5], who describe a dynamic program for finding the largest island in a set of  $n$  data points in  $O(n^3)$  time. Furthermore, Dumitrescu and Pach [14] study the worst-case cardinality of island partitions for different types of point sets. In our setting we want to use islands that capture the underlying point density (data distribution) well. Neither the largest island nor a partition with the fewest number of islands necessary support this quality criterion. Instead, we are using islands that cover their enclosed points well: data points are somewhat regularly distributed over the island and points which are not part of the island can easily be identified as such. Below we make this intuitive description more concrete.

We define a *bank* as a polyline whose vertices are data points (see Figures 5a and 6). We characterize the complexity of a bank via its number of *bends*. Specifically, each triple of consecutive points  $x, y, z$  on a bank defines a signed angle (in the interval  $(-\pi, \pi]$ ) between vectors  $\vec{xy}$  and  $\vec{yz}$ . A positive sign indicates a counter-clockwise turn. We define a *bend* in a bank as a maximal subsequence of data points such that the turning angles of all consecutive triples of data points within the sequence have the same sign. Figure 6 illustrates banks with lower or higher complexity (number of bends).

For SimpleSets we decided to limit the number of bends of a bank to two. Furthermore, we limit the sums of turning angles of our banks to 180 degrees and the maximum turning angle to 70 degrees. We chose these values after an exploratory evaluation on our datasets; other data values might serve other datasets better.

We now attempt to formalize the intuitive notion that islands and banks cover the data points well. Recall the quality criteria that we discussed in Section 2. There is a trade-off between strong continuation of shapes depicting the same set (C1) and the distortion of point density (C4) and the occlusion of a potential underlying visualization (C3). To point: a set visualization that draws only the data points results in no distortion and does not occlude a possible underlying visualization. However, there is little continuation between shapes that depict the same set. On the other hand, every visualization that adds any form of visual continuation will inevitably incur distortion and occlusion.

In SimpleSets, we quantify this trade-off as follows. We say that an island  $I$  is  $r$ -covered if its convex hull, including its interior, is covered

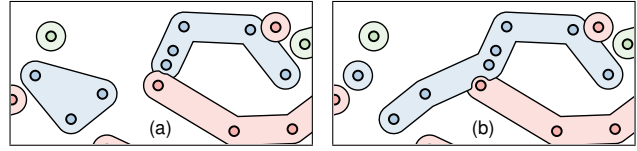


Fig. 6: Banks with a limited (a) or unlimited (b) number of bends.

by disks of radius  $r$  centered at data points in  $I$ . A bank is  $r$ -covered if the edge lengths of its polyline are bounded by  $2r$ . Computationally, we can verify whether an island  $I$  is  $r$ -covered for some  $r$  in  $O(|I| \log |I|)$  time. To do so, we compute a Voronoi diagram of  $I$ , and take the intersection with the convex hull of  $I$  (Figure 5b). The result is an arrangement consisting of Voronoi cells, some of which are clipped by the convex hull. Junctions where edges meet are called vertices. There are  $O(|I|)$  vertices, and for each we know the distance to its closest points in  $I$ . Island  $I$  is  $r$ -covered if, and only if, all vertices have distance at most  $r$  to their closest point in  $I$ .

The *cover radius*  $c(P)$  of a pattern  $P$  is the smallest  $r$  such that  $P$  is  $r$ -covered. If the cover radius limit is zero, then the only patterns are single data points. Conversely if the cover radius is not restricted, then any island or bank forms a pattern regardless of its distribution of data points. In between these two extremes, we find island and banks that form a spatial pattern in the sense that no location within the shape has distance (measured within the shape) larger than  $r$  to a data point.

In Section 4 we describe our algorithm that incrementally constructs a partition into islands and banks by merging patterns with ever increasing cover radius. We prefer patterns that are *regular* in the sense that all points inside the pattern are somewhat equally distributed and do not form clearly discernible sub-patterns (see Figure 7). This regularity can be quantified via the covering radius; in the next section we show how our preference can be incorporated in our incremental algorithm.



Fig. 7: Island  $P$  is not regular as it can be split into patterns  $P_1$  and  $P_2$  with  $\max(c(P_1), c(P_2))$  considerably smaller than  $c(P)$ .

Last but not least: we want our islands and banks to be clearly separated from other data points. After partitioning the input into patterns (islands and banks) our algorithm dilates all patterns with a *dilation radius*  $d_r$  and then draws the dilated patterns (see Figure 4). The patterns we create are disjoint by construction. However, we also want to avoid patterns that contain other points “too deep” inside their dilated shape. Naturally we cannot avoid input points that lie very close together. However, we will never create patterns that are too close to another point, which we quantify as half the dilation radius  $d_r$  (see Figure 8 left, note that points are drawn with a circle of size  $\approx d_r/3$ ). Patterns that contain other points within their dilated shape but at distance greater than  $d_r/2$  are admissible but not desirable (see Figure 8 middle); as with irregular islands we encourage our algorithm to not create them using a delay function. We quantify this so-called *intersection delay* using the area of overlap between a potential pattern and other existing patterns (see Figure 8 right). The technical details are described in Section 4.

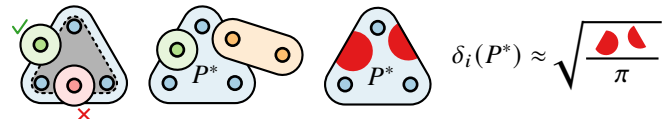


Fig. 8: Left: grey shows the region where no points of other patterns may lie. Middle-left: undesirable positioning of other patterns. Middle-right: intersection area of  $P^*$  with singleton points. Right: delay applied to events that would create  $P^*$ .

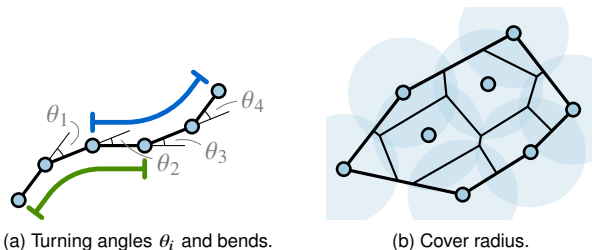


Fig. 5: Islands and banks.

### 3.2 Enclosing Shapes

The drawing algorithm of SimpleSets takes as input a set of disjoint, but possibly overlapping, patterns and computes an *enclosing shape* for each of them. When choosing the enclosing shape for SimpleSets, we considered four factors. First of all, we were of course looking for a *simple* shape. We consider a shape to be simple, if its boundary is smooth and consists of few geometric primitives. Second, we want the enclosing shape to be quite *similar* to the patterns we find in the data. Third, we want to use an enclosing shape that clearly *contains* the data points of the corresponding pattern, and fourth the enclosing shape should equally clearly *exclude* any data points that are not part of the pattern. In the previous subsection we already discussed how we can facilitate containment and exclusion when choosing patterns; here we discuss further design decisions that reinforce the results when drawing the enclosing shapes.

For SimpleSets we chose the Minkowski sum of a disk of the dilation radius  $r_d$  with the region of the pattern as the basis for our enclosing shape (see Figure 4). We refer to the resulting shapes as *dilated patterns*. Dilated patterns are simple according to our definition: their boundary is smooth and consists only of straight segments and circular arcs. Furthermore, by design dilated patterns are similar to their corresponding pattern and contain its data points in its interior (with a distance of at least  $d_r$  to the boundary). However, dilated patterns may contain data points in their interior (close to their boundary) which are not part of their pattern. This might occur because data points of different categories lie at a distances less than  $d_r$  or because we created an admissible but not desirable pattern (see the previous subsection). Hence, we need to find a consistent way to modify overlapping dilated patterns such that the set containment is clear for every data point.

**Overlapping dilated patterns.** Figure 9 shows three different methods to resolve overlapping dilated patterns. First of all, we could use a Voronoi diagram of the patterns and intersect each dilated pattern with the corresponding Voronoi cell (Figure 9b). However, such a drawing may contain very little visible area around a data point and hence can make it rather difficult to determine set containment for a data point. Furthermore, the Voronoi edges are parabolic arcs which make the enclosing shapes arguably less simple and less similar to their patterns.

Another option is to make use of an implicit third dimension and *stack* the enclosing shapes. Here we rely on the natural ability of the human viewer to complete contours that are partially occluded [31]. A stacked drawing provides us, in a sense, with more space to work with as a data point can be clearly outside an enclosing shape even if its position in the drawing is contained in the (implied) enclosing shape. See, for example, Figure 4 right: a blue point lies inside the dilated green island pattern, but since the blue bank shape has been visually stacked above the green enclosing shape, it is still clear that this point belongs to the blue bank only.

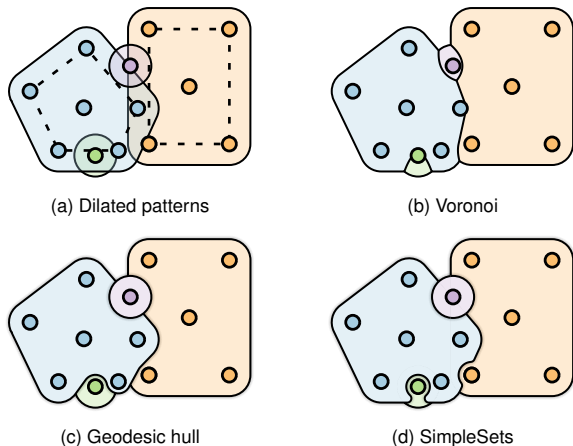


Fig. 9: Enclosing shapes that resolve overlaps.

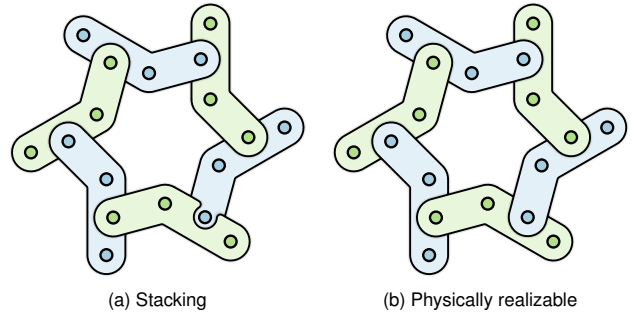


Fig. 10: Example instance where the physically realizable model allows better enclosing shapes than the stacking model.

When stacking dilated patterns we might still need to deform the boundary of an upper pattern to ensure the clear visibility of a point in a lower pattern. One option to do so are geodesic hulls, which are illustrated in Figure 9c. Because geodesic hulls use bitangents between data points, the enclosing shape might be deformed with respect to the dilated pattern even far away from overlaps. Since the bitangents can have any orientation, they also negatively impact the similarity of the enclosing shape to its pattern. For SimpleSets we decided to use smoothed circular cut-outs in the upper enclosing shapes to expose points in the lower shapes (see Figure 9d). Our enclosing shapes use only straight lines that are parallel to the boundary of their patterns, as well as circular arcs. This design, we believe, induces a clean look and feel and clearly shows the set containment of each data point.

**Stacking dilated patterns.** Since we decided to stack enclosing shapes, we need to determine a drawing order in the regions where the dilated patterns overlap each other. A simple approach would be to determine a total order on the shapes and stack them in this order. However, we choose the more powerful physically realizable model [9] that has no global stacking order, but instead only stacks shapes locally (Figure 10). More precisely, we allow an enclosing shape to be drawn on top of another in one area, but below that same other shape in a different area, as long as this can be achieved by “bending” the shapes in the implied third dimension; cutting shapes is not allowed. Physically realizable stacking is heavily utilized in the SimpleSets drawings of LineSets and Bubble Sets in Figure 6 of the supplementary material.

When deciding on the best local stacking orders, we follow a few simple principles, based on the assumption that humans can visually complete interrupted or modified line segments better than circular arcs (see Figure 11). First and foremost, we prefer orders which do not require us to introduce cut-outs to avoid points. If we cannot avoid adding a cut-out, then we prefer to modify a line segment over modifying a circular arc. Finally, if we need to cover a piece of an enclosing shape, then we prefer to cover a line segment over covering a circular arc. In Section 5 we explain how we capture these preferences in our drawing algorithm.

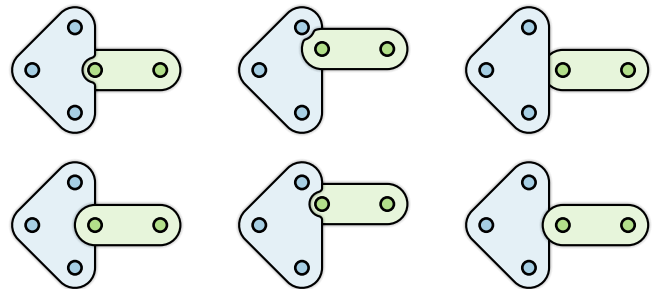


Fig. 11: Stacking preferences, preferred stacking at the bottom. Left: avoids modification of shape; middle: avoids modification of circular arc; right: avoids covering of circular arc.

## 4 PARTITIONING ALGORITHM

SimpleSets uses a greedy incremental clustering algorithm to partition categorical point data into disjoint and non-overlapping islands and banks. We initialize our algorithm with a partition where each data point forms a separate pattern; afterwards we iteratively merge patterns. Whenever our algorithm merges two source patterns  $P_1$  and  $P_2$  into a target pattern  $P^*$ , we ensure that the data points and regions of  $P_1$  and  $P_2$  are subsets of the data points and region of  $P^*$ . As a result, the size of the regions covered by patterns monotonically increases as our algorithm proceeds. We store all intermediate partitions; the sequence of partitions forms a topological filtration and can also be interpreted as a hierarchical agglomerative clustering on the input points.

Our algorithm performs a discrete event simulation, starting at time  $t = 0$  with the initial disjoint patterns. As  $t$  increases, we merge patterns in the partition into larger patterns. We maintain the following invariant:

- The patterns in the partition are disjoint and do not overlap.
- All patterns are  $t$ -covered.

Thus the time parameter  $t$  is a single intuitive parameter that allows the user to select the scale at which they wish to define patterns. In Figure 5 of the supplementary material we illustrate the influence of  $t$  via four SimpleSets visualizations of one dataset that are constructed from partitions at different times in the sequence.

There is only one type of event in our simulation: the *merge event*. A merge event is associated with two source patterns  $P_1$  and  $P_2$  to be merged, the target pattern  $P^*$  that is the result of the merge, and the time  $t$  at which the merge would take place. We maintain the events in a priority queue on the event times and handle them in order of priority; two events that occur at the same time are handled in arbitrary order.

Below we describe in detail how we create and handle events in our discrete event simulation. In principle, the time  $t$  of a specific event corresponds directly to the covering radius that allows the corresponding pattern to be formed. However, as discussed in Section 3, we prefer regular islands, which contain points that are somewhat equally distributed, over patterns that exhibit clear sub-patterns (see Figure 12). Hence we add a processing delay to those events that would create islands which are not regular. This delay increases the time at which the event would occur and thereby increases the chance that the corresponding pattern is never formed, since other merge events might already have used a source pattern, or have blocked the target pattern, by the time the delayed event occurs.

We also prefer patterns that are well-separated from other points. Patterns that are too close to another point (distance less than half the dilation radius  $r_d$ ) are never created by our algorithm, since the dilated pattern would enclose that unrelated point too much. Patterns that keep a distance of less than  $r_d$  to other points result in dilated patterns that still enclose those unrelated points, but less deeply. As discussed in Section 3, these patterns are admissible but not desirable; we hence add a delay to the events that would create them.

**Creating events.** Our algorithm creates one or multiple events for two source patterns  $P_1$  and  $P_2$  as follows. If either  $P_1$  or  $P_2$  is an island, then we create one event for them to merge into an island  $P^*$  which is the convex hull of the data points contained in  $P_1$  and  $P_2$ . To maintain

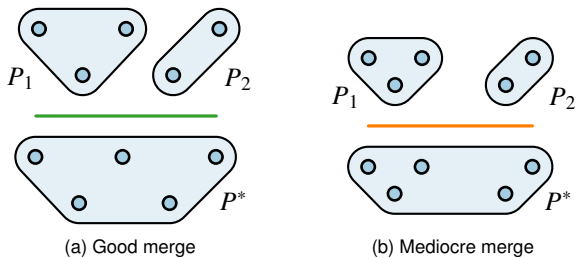


Fig. 12: The target patterns  $P^*$  in (a) and (b) have identical cover radius  $c(P^*)$ . But the pattern in (b) is less regular and has more pronounced sub-patterns. Our discrete event simulation delays merge (b).

the invariant of our algorithm and ensure that the sequence of patterns is indeed a filtration, we do not allow islands to merge into a bank.

If both  $P_1$  and  $P_2$  are banks, then we create up to four events that correspond to the four ways in which we can connect the two ends each of  $P_1$  and  $P_2$ . If the polyline corresponding to a specific connection is a bank according to our definition, then it becomes a target pattern  $P^*$ .

As discussed above, a priori the time  $t$  associated with an event is the cover radius  $c(P^*)$  of  $P^*$ . However, a less desirable merge will incur additional delays to reduce the chance that it is actually executed. If  $P_1$  and  $P_2$  both have a small cover radius then we delay merging them into a pattern  $P^*$  with a large cover radius (Figure 12), since that merge would create a pattern which is not regular. Such a merge is delayed by a *regularity delay* of  $\delta_r(P_1, P_2, P^*) = c(P^*) - \max(c(P_1), c(P_2))$ . If both  $P_1$  and  $P_2$  are single data points then we add no delay. Next we measure the overlap (the area of intersection) of the target pattern  $P^*$  with all dilated data points. Let  $D_U$  denote the union of dilated data points and let  $D_1, D_2$  and  $D^*$  denote the dilated versions of  $P_1, P_2$  and  $P^*$ . We first compute the set  $A = (D^* \setminus (D_1 \cup D_2)) \cap D_U$  of overlap between  $D^*$  and the parts of dilated points that are neither in  $D_1$  nor in  $D_2$ . Then we use the area  $a$  of this overlap  $A$  to set the *intersection delay*  $\delta_i(P^*) = \sqrt{a/\pi}$  (see Figure 8 right).

**Initialization.** We initialize the algorithm with a partition where each data point is a pattern. During initialization, we create events for every pair of distinct data points of the same category.

**Handling events.** When handling an event, we first need to check if the corresponding merge is possible given the current partition into patterns. If either of the source patterns  $P_1$  or  $P_2$  are not part of the current partition, then the merge is not possible and we simply skip the event. Similarly, if the target pattern  $P^*$  overlaps any pattern in the current partition, other than  $P_1$  and  $P_2$ , the merge is not possible and the event is skipped. If the merge is possible, then we remove patterns  $P_1$  and  $P_2$  from the current partition and replace them by pattern  $P^*$ . Then we create events for the new pattern  $P^*$  and each of the other patterns of the same category in the current partition. If the current event was processed with a regularity or an intersection delay, then the newly created events might lie in the past. In this case they are immediately processed by our algorithm.

**Running time.** We now analyse the running time of our partitioning algorithm. Let  $n$  denote the number of data points. We can compute the regularity and intersection delays in  $O(n^2)$  time and, thus, create a new event in  $O(n^2)$  time. Furthermore, we can decide in  $O(n^2)$  time whether an event must be skipped. We initially create  $O(n^2)$  events in  $O(n^4)$  time. Our discrete event simulation modifies the partition at most  $n - 1$  times, since each merge reduces the number of patterns in the partition by one. Therefore, at most  $n - 1$  events result in actual merges, creating  $O(n)$  new events each time. Thus, we handle in total  $O(n^2)$  events, each of which takes  $O(n^2)$  time. Insertions and deletions in the priority queue take  $O(\log n)$  time each. Hence, the total running time of the partitioning algorithm is  $O(n^4)$ .

A running time of  $O(n^4)$  is comparatively high and sounds forbidding in practice, however, set visualizations are rarely applied to sets with thousands of data points at predefined locations. Our implementation created the SimpleSets visualization in Figure 2 in 232 milliseconds on a laptop with an Intel(R) Core(TM) i7-11800H 2.30 GHz processor. The intersection delay has a large impact on the running time but little impact on the partitions in practice; omitting it speeds up the computation to 58 milliseconds. In Figure 15 we plot the running times for the other examples discussed in this paper: computing SimpleSets for datasets with 500 data points still takes only about 1 second.

**Choosing a partition.** As mentioned above, our partitioning algorithm returns a sequence of partitions and the user can choose a suitable partition via the time parameter  $t$ . In our experiments, we found that partitions in time range  $[2r_d, 6r_d]$ , where  $r_d$  denotes the dilation radius, capture patterns well and provide a good trade-off between connecting points and covering space. Figure 5 of the supplementary material uses partitions at times  $2.5r_d, 3.5r_d, 4.5r_d$ , and  $6r_d$ .

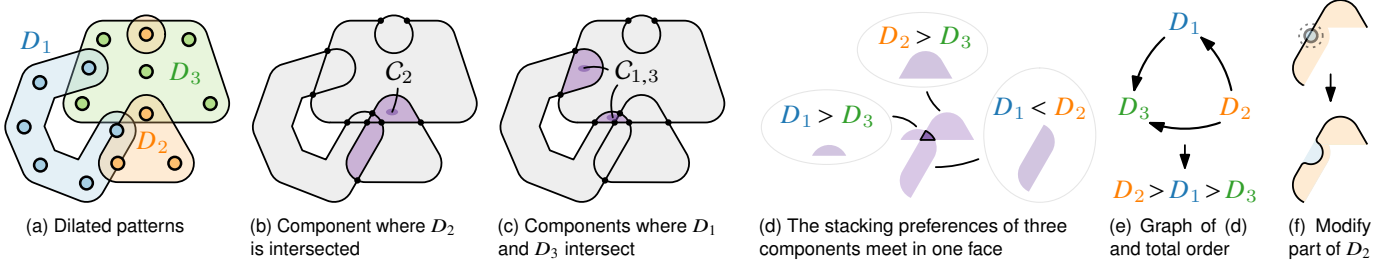


Fig. 13: An overview of the drawing algorithm. See Figure 4 for the input patterns and the final rendering. In this example, the hypergraph  $G$  consists of only one non-singleton hyperedge, shown in (d), as there is only one face in the arrangement that is contained in at least three dilated patterns.

## 5 DRAWING ALGORITHM

The output of the partitioning algorithm described in Section 4 is a set of disjoint patterns that do not overlap. Our drawing algorithm can also handle sets of disjoint, but possibly overlapping patterns. In the supplementary material we illustrate this by rendering patterns from the state-of-the-art with our drawing algorithm. Disjoint but possibly overlapping patterns are, therefore, the input that we assume for this section. Figure 13 and Algorithm 1 summarize our drawing algorithm.

In a first step, we dilate each pattern using the Minkowski sum with a disk of radius  $d_r$  (see Figure 4). We denote the dilated patterns by  $D_1, \dots, D_m$ . Even if the patterns do not overlap, the dilated patterns might (see Figure 13a). Wherever two or more dilated patterns overlap, we have to determine their drawing order. In Section 3 we discussed our preferences for stacking two enclosing shapes; in this section we explain how to combine such pairwise-preferences into local stacking orders for a set of enclosing shapes.

Once we have computed local stacking orders, we might have to modify the boundary curves of the upper shapes to ensure that data points in lower shapes remain visible. In Section 3 we discussed design alternatives for modified upper shapes and settled on circular cut-outs in otherwise straight-line convex hulls for SimpleSets. Computing these cut-outs is conceptually simple; however, several edge cases require additional care. We attempt to convey the major considerations in this section and refer the reader to the supplementary material for details.

**Computing stacking orders.** We formally approach this problem as follows, see Figure 13 for illustrations. As a first step, we build the arrangement  $\mathcal{A}$  [6] induced by the dilated patterns. Every face  $f$  of  $\mathcal{A}$  stores a set  $D(f)$  of dilated patterns that contain the face. Let  $C_{i,j}$  denote the set of connected components of faces  $f$  in  $\mathcal{A}$  such that  $D_i$  and  $D_j$  are part of  $D(f)$ . Let  $C_i$  be the set of connected components of faces  $f$  in  $\mathcal{A}$  where  $D_i \in D(f)$  and  $|D(f)| \geq 2$ . For each pair of dilated patterns  $D_i, D_j$  and component  $C \in C_{i,j}$  we create a relation  $R_{i,j,C}$  that takes a value in  $\{<, =, >\}$ , which indicates the order in which  $D_i$  and  $D_j$  are stacked in component  $C$ .

Initially  $R_{i,j,C}$  contains the stacking preference according to the simple rules described in Section 3. If no rule applies then there is no preference, indicated by  $=$ . After this initial step, every face  $f$  in the arrangement contains a set of relations  $R(f)$ . For our drawing, we need for every face  $f$  a stacking order on  $D(f)$  such that the orders of all faces together form a physically realizable drawing. To achieve this, we set each relation  $R_{i,j,C}$  to  $<$  or  $>$ , and derive a stacking order in  $f$  from  $R(f)$ . Faces where three or more dilated patterns coincide contain more than one relation and impose constraints because the relations in a face should not form a cyclic dependency (Figure 13d).

We capture these constraints in a hypergraph on the set of all relations  $R_{i,j,C}$ . Each hyperedge will be a set of relations that meet in a face and should not form a cyclic dependency. If  $R(f) \subseteq R(f')$  for distinct faces  $f$  and  $f'$ , then  $f$  can inherit the ordering computed for  $f'$ . Hence, the edges of the hypergraph are  $E = \{R(f) \mid R(f) \text{ is maximal}\}$ . Consider a hyperedge  $e \in E$ . We compute a total ordering of the dilated patterns that are part of the relations of this hyperedge by constructing a graph  $G_e$ . The vertices of  $G_e$  are the set of all indices of dilated patterns that are present in the relations stored in  $e$ . An edge is present in  $G_e$ , directed from  $i$  to  $j$ , if a relation indicates that  $D_i$  should be stacked on

---

### Algorithm 1: Drawing enclosing shapes

---

**Input:** Patterns  $P_1, \dots, P_m$

**Output:** A drawing of  $P_1, \dots, P_m$  using enclosing shapes

$D_1, \dots, D_m \leftarrow$  dilate patterns  $P_1, \dots, P_m$

$\mathcal{A} \leftarrow$  arrangement of  $D_1, \dots, D_m$

**for**  $1 \leq i < j \leq m$  **do**

**for**  $C \in C_{i,j}$  **do**

        create a relation  $R_{i,j,C} \in \{<, =, >\}$  and store a reference to  $R_{i,j,C}$  in each face  $f \in C$  in set  $R(f)$   
        compute the stacking preference of  $i$  and  $j$  in  $C$  and set  $R_{i,j,C}$  accordingly

$G(V, E) \leftarrow$  hypergraph where relations  $R_{i,j,C}$  form  $V$  and  $E$  consists of maximal sets of relations present in a face

**for**  $e \in E$  **do**

    determine a stacking order for  $e$  and set each  $R_{i,j,C} \in e$  to  $<$  or  $>$  appropriately

**for** face  $f$  in  $\mathcal{A}$  **do**

    determine a stacking order in  $f$  based on  $R(f)$

**for**  $1 \leq i \leq m$  **do**

**for**  $C \in C_i$  **do**

        modify parts of  $D_i$  in  $C$  such that data points of shapes below  $i$  in the order stored in  $f \in C$  are exposed

draw each face of  $\mathcal{A}$

---

top of  $D_j$ . If  $G_e$  has no cycles then we use a topological order of  $G_e$  and set the relations  $R_{i,j,C}$  to match this total order (Figure 13e).

If  $G_e$  has cycles, then we remove them by flipping edges. We want to minimize the number of flipped edges, since they correspond to drawing orders that go against our rules. The corresponding computational problem is challenging, but brute-force solutions are computationally feasible for standard set visualizations. However, in all our experiments with SimpleSets we have not encountered a single example where  $G_e$  has a cycle. Stronger yet, in all our SimpleSets examples there is a total stacking order of enclosing shapes which satisfies our rules.

**Curve modification.** Once the local stacking orders have been determined, we modify the dilated patterns where needed to expose data points inside lower shapes. This process is summarized in Figure 14. Consider an arbitrary component  $C \in C_i$  for some  $i$ . Let  $A$  be the set of patterns that are below  $D_i$  in the stacking order in the faces of  $C$ . We expose data points of patterns in  $A$  by modifying  $D_i$ .

Our algorithm aims to keep a disk of radius  $r_c = 5/8 \cdot r_d$  centered at each element visible. To this end, we place exclusion (red) disks of radius zero at each data point of each pattern in  $A$ . We also place inclusion (green) disks of radius zero at each data point of  $P_i$ . Then we grow the disks at a uniform rate. If two disks of different color collide then we stop their growth. We also stop the growth of a red disk when it reaches  $r_c$  and the growth of a green disk when it reaches  $r_d$ .

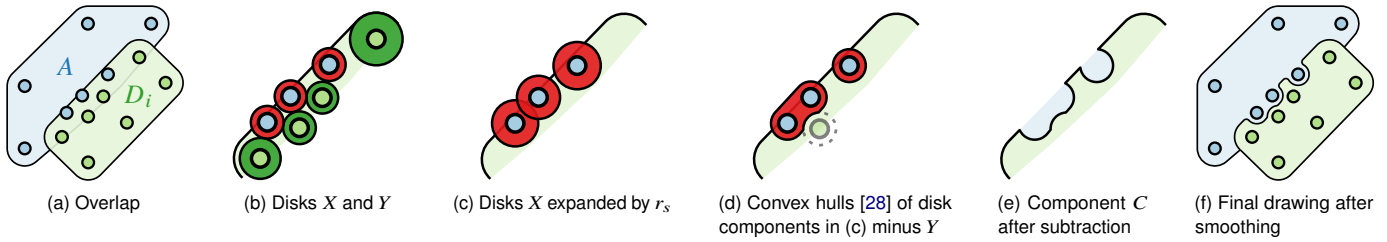


Fig. 14: Modification of dilated patterns to expose points beneath. Figures (b)–(e) show a closeup of the component  $C$  in (a) and (f).

The result of this growth process is a set  $X$  of exclusion disks and a set  $Y$  of inclusion disks such that any disk in  $X$  is disjoint from those in  $Y$  and vice versa (Figure 14b). We now cut the disks in  $X$  out of  $D_i$  to expose the corresponding points and smooth the result using the Minkowski sum and difference with a disk of radius  $r_s = r_d/5$ . When disks in  $X$  are sufficiently close (Figure 14c) then we cut out their convex hull [28] instead. We ensure that the disks in  $Y$  lie in the final pattern by not cutting them out (Figure 14d).

**Running time.** The arrangement has complexity  $O(n^2)$  and can be computed in  $O(n^2 \log n)$  time using a sweep-line algorithm. The other parts of the algorithm run in  $O(n^2)$  time. The partitioning algorithm clearly dominates the running time.

## 6 EVALUATION

We evaluate SimpleSets based on the four criteria of Dinkla et al. [13] stated in Section 2. Below, we introduce measures to capture these criteria, allowing for a quantitative analysis. We compare SimpleSets to VPF\*14 [30] and ClusterSets [18], as these techniques also use multiple shapes to represent each set. We create and measure two versions of VPF\*14: VPF\*14+ where each vertex has a large radius of influence on the potential field, and VPF\*14- where each vertex has a medium radius of influence. VPF\*14+ best matches the visualizations in the paper that introduced the method; VPF\*14- uses a 25% smaller radius.

**Implementation.** A prototype implementation of SimpleSets written in Kotlin is available at [github.com/tue-alga/SimpleSets](https://github.com/tue-alga/SimpleSets). For easy access, the repository links to a web-based implementation created by compiling the Kotlin code to JavaScript. The running time of the JVM implementation is shown in Figure 15. All SimpleSets visualizations in this paper were generated by this implementation in seconds. The drop-shadows present in some of the figures were added manually.

**Data.** We use the following three datasets in our evaluation. These datasets are available in the linked GitHub repository in the folder `src/commonMain/resources/example-input`.

**Mills** Mills in the vicinity of Leeuwarden, The Netherlands (Figure 1). The dataset consists of 55 mills of four types.

**NYC** Points of interest located in lower Manhattan (Figure 2). The dataset consists of 96 points of three types.

**HDN** Vertices of an embedded graph of disorders, from the human disease network constructed by Goh et al. [19]. Figure 16 shows an extract. The full dataset consists of 516 vertices (disorders) of twenty-one disorder classes. We follow Vihrovs et al. [30] in using the graph layout of a poster by Bastian and Heymann archived at

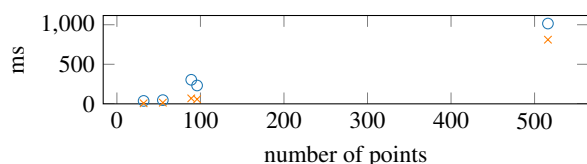


Fig. 15: Total running time of SimpleSets in milliseconds on our datasets. Crosses show the running time without intersection delay.

[https://web.archive.org/web/20121116145141/http://diseasome.eu/data/diseasome\\_poster.pdf](https://web.archive.org/web/20121116145141/http://diseasome.eu/data/diseasome_poster.pdf).

**Measures.** We measure criterion C1, cognitive load, by measuring various aspects of shape complexity. The complexity of shapes is difficult to measure. A wide range of measures have been proposed and this is still an active research topic [4, 7, 12, 26]. We require measures that work for both the smooth shapes that are present in the VPF\*14 visualization and the more polygonal, but still smooth, shapes of ClusterSets and SimpleSets. We use the following four measures; we also provide citations to relevant literature.

**Inflections [4, 13]** For each shape, determine the number of inflection points. An inflection point is where the curvature of a shape changes sign. We measure the total number of inflection points present in a visualization.

**Perimeter ratio [7]** The ratio of the perimeter of a shape to the perimeter of the convex hull of the shape. We measure the average and maximum ratios among all shapes.

**Area ratio [7]** The ratio of the area of a shape to the area of the convex hull of the shape. We measure the average and maximum ratios among all shapes.

**Curvature [23]** The total absolute curvature of the shape in radians. As all visualizations use closed shapes this is at least  $2\pi$  for each shape; therefore, for better comparison we subtract  $2\pi$ . We measure the average and maximum value among all shapes.

To measure the number of inflection points and the total absolute curvature we sample equidistant points on the contours of the shapes (Figure 7 of supplementary material). Holes of shapes also contribute to the number of inflections, the perimeter, and curvature of the shape (and negatively to its area). The number of shapes (the measure we use for C2), also readily affects cognitive load, and should be taken into account as convex shapes score perfectly on the above measures.

We measure C2, strong continuation, by the number of shapes. This factor is evidently not the only factor affecting the visual continuation of sets; however, this is an attribute that is straightforward to measure and clearly correlates with continuation for the three visualizations we are comparing.

The third criterion C3 is straightforward to measure: we measure the area covered by the visualization. To normalize, we divide by the area of the bounding box of the input points and we present the result as a percentage value. Note that the value could exceed 100% as shapes may lie partially outside the bounding box of the input points.

Lastly, criterion C4: distortion of point position and density. We use the following two measures.

**Density distortion** The absolute difference, per set, between the fraction of input points that belong to this set, and the fraction of area covered by shapes of this set compared to the total area covered by the visualization. We measure the average and maximum value.

**Cover radius** The cover radius of a pattern captures parts of a pattern that are far away from the input points it represents. We measure the average and maximum cover radius, over all patterns. Note that the cover radius, opposed to all other values we measure, is scale-dependent and, therefore, makes sense only in comparison with visualizations on the same dataset.



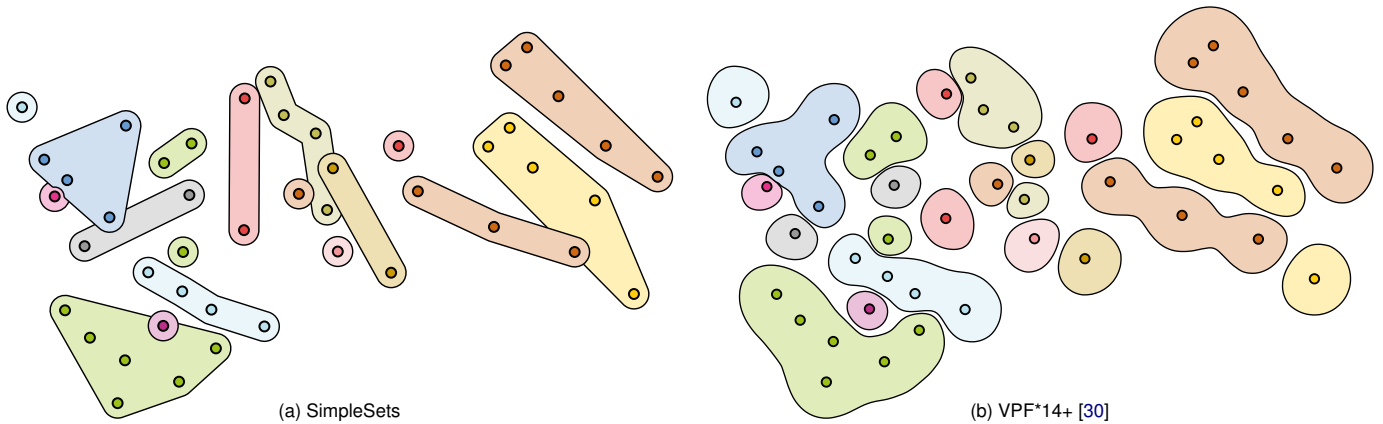


Fig. 16: Part of the human disease network (HDN) dataset by Goh et al. [19]

Table 1: Quantitative comparison of SimpleSets, Vihrovs et al., and ClusterSets on three datasets. Where applicable, measures have the format *avg / max*, showing both the average and maximum among all colors or shapes as appropriate. Lower values are better.

		C1			C1, C2	C3	C4		
		Inflections	Perimeter ratio	Area ratio	Curvature	Shapes	Covered area	Density distortion	Cover radius
Mills	SimpleSets	8	1.01 / 1.25	1.07 / 2.04	0.72 / 5.86	26	19.4%	6.11% / 11.7%	5.00 / 16.4
	VPF*14+	45	1.03 / 1.20	1.06 / 1.33	2.61 / 13.9	25	51.1%	1.84% / 3.48%	2.65 / 13.3
	VPF*14-	22	1.01 / 1.09	1.02 / 1.26	0.87 / 7.41	35	32.7%	1.68% / 3.27%	1.84 / 11.2
	ClusterSets	18	1.11 / 1.40	1.48 / 2.67	5.91 / 26.3	8	48.5%	8.05% / 13.7%	23.2 / 37.2
NYC	SimpleSets	16	1.01 / 1.11	1.07 / 1.87	1.03 / 7.39	36	25.3%	0.89% / 2.07%	6.64 / 23.4
	VPF*14+	60	1.03 / 1.20	1.07 / 1.51	3.10 / 17.1	36	55.2%	1.17% / 1.75%	4.65 / 18.5
	VPF*14-	60	1.03 / 1.16	1.07 / 1.41	2.66 / 13.3	43	39.8%	0.49% / 0.74%	3.31 / 18.2
	ClusterSets	29	1.19 / 1.74	1.57 / 3.10	8.29 / 30.2	10	61.7%	6.87% / 10.0%	28.8 / 68.1
HDN	SimpleSets	36	1.00 / 1.07	1.02 / 1.78	0.27 / 9.38	229	21.6%	1.09% / 8.37%	12.5 / 78.2
	VPF*14+	383	1.02 / 2.07	1.04 / 1.63	1.79 / 110	258	36.2%	0.54% / 3.13%	4.73 / 44.4
	VPF*14-	406	1.02 / 1.89	1.04 / 1.94	1.92 / 70.5	294	26.5%	0.37% / 1.92%	4.49 / 39.8
	ClusterSets	134	1.08 / 2.35	1.20 / 4.97	3.30 / 88.0	163	31.7%	1.87% / 10.9%	30.2 / 234

**Results and discussion.** For the visualizations, refer to Figure 2 (NYC) and supplementary material Figure 8 (Mills), Figure 9 (NYC, VPF\*14-) and Figures 10–13 (HDN). Table 1 shows the measurements of these visualizations; for all measures, lower values indicate better performance.

SimpleSets scores better than ClusterSets on all measures except the number of shapes. The individual shapes of SimpleSets are simpler. As SimpleSets uses more shapes, though, it is unclear whether the cognitive load is lower when viewing a SimpleSets visualization compared to a ClusterSets visualization.

Comparing SimpleSets to VPF\*14, we see that the shapes of VPF\*14 are irregular and cover considerably more area. VPF\*14+ uses roughly the same number of shapes as SimpleSets; VPF\*14- uses more shapes but covers less area than VPF\*14+. Both versions of VPF\*14 generally score better on our quantitative measures of C4. The usage of a potential field by VPF\*14 explains these differences. In VPF\*14 each point has an area of influence and shapes merge only when their areas of influence touch. In contrast, in SimpleSets two singleton patterns may use significantly less area than a bank joining the two points. See for example the two red points in Figure 16 that are part of one large vertical bank in SimpleSets, but are separate shapes in VPF\*14. Though not captured in the quantitative measures, VPF\*14 does distort point position and density (C4) in ways that SimpleSets does not. In particular, the shapes used by VPF\*14 depend on the distribution of points around it, which means the same pattern of points may be drawn quite differently depending on its surroundings. This is most clear in the shapes enclosing single points: they vary greatly in size, and the data point often lies away from the centroid of the shape. In contrast, SimpleSets encloses single points with circles with the data point at the center of the circle. The approach of Vihrovs et al. does have its advantages: the bottom-left green shape in Figure 16b in their visual-

ization nicely follows the distribution of points and avoids creating an intersection with the shape enclosing the magenta point. However, in general it introduces unnecessary complexity and distortion.

## 7 FUTURE WORK

The SimpleSets partitioning algorithm uses two delays, which work well for our datasets. It would be interesting to measure the effect these delays have; furthermore, one could consider additional delays such as a delay that penalizes merges that add an inflection to a bank. In addition, SimpleSets relies on color to convey the category to which a point or shape belongs. Therefore, they can clearly depict only a limited number of categories. The use of additional visual attributes to encode the categories could mitigate this limitation.

When applying the SimpleSets drawing algorithm to large overlapping patterns, such as those of LineSets and Bubble Sets (see supplementary material), there are intersections where no preference rules apply, which leads to arbitrary “weaving” of the drawing. In this context it may be interesting to investigate the minimization of weaving [16]. We note that a user study might be insightful and could settle lingering questions that our quantitative evaluation could not answer regarding the effectiveness of SimpleSets compared to other visualizations.

We designed SimpleSets to visualize points that each belong to one category. An interesting direction for future work would be to adapt SimpleSets to visualize points belonging to multiple categories. Though several parts readily extend, various questions on visual design and algorithms need to be answered on the way to an automated solution.

We close with a general question for set visualizations: how can we handle datasets that contain both very dense and quite sparse areas? Such datasets are abundant in practice, which limits the practical application of current set visualizations for categorical point data.

## SUPPLEMENTARY MATERIALS

The supplementary material contains a detailed description of the process for computing cutouts and eight supplementary figures. Figure 5 shows four SimpleSets partitions for different time parameters. Figure 6 illustrates that any set of disjoint patterns can be drawn with the SimpleSets drawing algorithm. Figure 7 shows the equidistant point sampling used to measure inflections and curvature for the quantitative analysis. The remaining figures are outputs not present in the paper that were analyzed in the quantitative evaluation.

The code is available at [github.com/tue-alga/SimpleSets](https://github.com/tue-alga/SimpleSets) and has been archived at [doi:10.5281/zenodo.12784670](https://doi.org/10.5281/zenodo.12784670).

## FIGURE CREDITS

Figure 2b is based on Figure 9c of the paper by Vihrov et al. [30], but has been generated using our implementation of their method. Figures 2c–2g are adapted from the paper by Geiger et al. [18]. The backdrop map in Figure 2 is from [openstreetmap.org](https://openstreetmap.org).

## REFERENCES

- [1] H. A. Akitaya, M. Löffler, and C. D. Tóth. Multi-colored spanning graphs. *Theoretical Computer Science*, 833:11–25, 2020. doi: [10.1016/j.tcs.2020.04.022](https://doi.org/10.1016/j.tcs.2020.04.022) 3
- [2] B. Alper, N. Henry Riche, G. A. Ramos, and M. Czerwinski. Design study of LineSets, a novel set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2259–2267, 2011. doi: [10.1109/TVCG.2011.186](https://doi.org/10.1109/TVCG.2011.186) 2, 3, 12
- [3] B. Alsallakh, L. Micaleff, W. Aigner, H. Hauser, S. Miksch, and P. J. Rodgers. The state-of-the-art of set visualization. *Computer Graphics Forum*, 35(1):234–260, 2016. doi: [10.1111/CGF.12722](https://doi.org/10.1111/CGF.12722) 2, 3
- [4] F. Attneave. Physical determinants of the judged complexity of shapes. *Journal of Experimental Psychology*, 53(4):221, 1957. doi: [10.1037/h0043921](https://doi.org/10.1037/h0043921) 8
- [5] C. Bautista-Santiago, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urutia, and I. Ventura. Computing optimal islands. *Operations Research Letters*, 39(4):246–251, 2011. doi: [10.1016/j.orl.2011.04.008](https://doi.org/10.1016/j.orl.2011.04.008) 4
- [6] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars. *Computational geometry: algorithms and applications*. Springer, third ed., 2008. doi: [10.1007/978-3-540-77974-2\\_3\\_7](https://doi.org/10.1007/978-3-540-77974-2_3_7)
- [7] T. Brinkhoff, H. Kriegel, R. Schneider, and A. Braun. Measuring the complexity of polygonal objects. In *Proc. 3rd ACM International Workshop on Advances in Geographic Information Systems*, p. 109. ACM, 1995. 8
- [8] H. Byelas and A. C. Telea. Towards realism in drawing areas of interest on architecture diagrams. *Journal of Visual Languages & Computing*, 20(2):110–128, 2009. doi: [10.1016/j.jvlc.2008.09.001](https://doi.org/10.1016/j.jvlc.2008.09.001) 3
- [9] S. Cabello, H. J. Haverkort, M. J. van Kreveld, and B. Speckmann. Algorithmic aspects of proportional symbol maps. *Algorithmica*, 58(3):543–565, 2010. doi: [10.1007/s00453-009-9281-8](https://doi.org/10.1007/s00453-009-9281-8) 5
- [10] T. Castermans, M. van Garderen, W. Meulemans, M. Nöllenburg, and X. Yuan. Short plane supports for spatial hypergraphs. *Journal of Graph Algorithms and Applications*, 23(3):463–498, 2019. doi: [10.7155/JGAA.00499](https://doi.org/10.7155/JGAA.00499) 3
- [11] C. Collins, G. Penn, and S. Cappendale. Bubble Sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009. doi: [10.1109/TVCG.2009.122](https://doi.org/10.1109/TVCG.2009.122) 2, 3
- [12] L. Dai, K. Zhang, X. S. Zheng, R. R. Martin, Y. Li, and J. Yu. Visual complexity of shapes: a hierarchical perceptual learning model. *Visual Computer*, 38(2):419–432, 2022. doi: [10.1007/S00371-020-02023-Z](https://doi.org/10.1007/S00371-020-02023-Z) 8
- [13] K. Dinkla, M. J. van Kreveld, B. Speckmann, and M. A. Westenberg. Kelp Diagrams: Point set membership visualization. *Computer Graphics Forum*, 31(3pt1):875–884, 2012. doi: [10.1111/j.1467-8659.2012.03080.x](https://doi.org/10.1111/j.1467-8659.2012.03080.x) 2, 3, 8
- [14] A. Dumitrescu and J. Pach. Partitioning colored point sets into monochromatic parts. *International Journal of Computational Geometry & Applications*, 12(5):401–412, 2002. doi: [10.1142/S0218195902000943](https://doi.org/10.1142/S0218195902000943) 4
- [15] A. Efrat, Y. Hu, S. G. Kobourov, and S. Pupyrev. MapSets: Visualizing embedded and clustered graphs. *Journal of Graph Algorithms and Applications*, 19(2):571–593, 2015. doi: [10.7155/jgaa.00364](https://doi.org/10.7155/jgaa.00364) 2, 3
- [16] D. Eppstein, M. J. van Kreveld, E. Mumford, and B. Speckmann. Edges and switches, tunnels and bridges. *Computational Geometry*, 42(8):790–802, 2009. doi: [10.1016/J.COMGEO.2008.05.005](https://doi.org/10.1016/J.COMGEO.2008.05.005) 9
- [17] E. R. Gansner, Y. Hu, and S. G. Kobourov. GMap: Visualizing graphs and clusters as maps. In *Proc. Pacific Visualization Symposium (PacificVis)*, pp. 201–208. IEEE, 2010. doi: [10.1109/PACIFICVIS.2010.5429590](https://doi.org/10.1109/PACIFICVIS.2010.5429590) 3
- [18] J. Geiger, S. Cornelsen, J. Haunert, P. Kindermann, T. Mchedlidze, M. Nöllenburg, Y. Okamoto, and A. Wolff. ClusterSets: Optimizing planar clusters in categorical point data. *Computer Graphics Forum*, 40(3):471–481, 2021. doi: [10.1111/cgf.14322](https://doi.org/10.1111/cgf.14322) 2, 3, 8
- [19] K.-I. Goh, M. E. Cusick, D. Valle, B. Childs, M. Vidal, and A.-L. Barabási. The human disease network. *Proceedings of the National Academy of Sciences*, 104(21):8685–8690, 2007. doi: [10.1073/pnas.0701361104](https://doi.org/10.1073/pnas.0701361104) 8, 9
- [20] J. Heer and d. boyd. Vizster: Visualizing online social networks. In *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 32–39. IEEE, 2005. doi: [10.1109/INFVIS.2005.1532126](https://doi.org/10.1109/INFVIS.2005.1532126) 2
- [21] N. Henry Riche and T. Dwyer. Untangling Euler diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1090–1099, 2010. doi: [10.1109/TVCG.2010.210](https://doi.org/10.1109/TVCG.2010.210) 2
- [22] F. Hurtado, M. Korman, M. J. van Kreveld, M. Löffler, V. Sacristán, A. Shioura, R. I. Silveira, B. Speckmann, and T. Tokuyama. Colored spanning graphs for set visualization. *Computational Geometry*, 68:262–276, 2018. doi: [10.1016/J.COMGEO.2017.06.006](https://doi.org/10.1016/J.COMGEO.2017.06.006) 3
- [23] T. Matsumoto, K. Sato, Y. Matsuoka, and T. Kato. Quantification of “complexity” in curved surface shape using total absolute curvature. *Computers & Graphics*, 78:108–115, 2019. 8
- [24] W. Meulemans, N. Henry Riche, B. Speckmann, B. Alper, and T. Dwyer. KelpFusion: A hybrid set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 19(11):1846–1858, 2013. doi: [10.1109/TVCG.2013.76](https://doi.org/10.1109/TVCG.2013.76) 2, 3, 12
- [25] P. Paetzold, R. Kehlbeck, H. Strobel, Y. Xue, S. Storandt, and O. Deussen. RectEuler: Visualizing intersecting sets using rectangles. *Computer Graphics Forum*, 42(3):87–98, 2023. doi: [10.1111/CGF.14814](https://doi.org/10.1111/CGF.14814) 2
- [26] D. L. Page, A. F. Koschan, S. R. Sukumar, B. Roui-Abidi, and M. A. Abidi. Shape analysis algorithm based on information theory. In *Proc. International Conference on Image Processing*, pp. 229–232. IEEE, 2003. doi: [10.1109/ICIP.2003.1246940](https://doi.org/10.1109/ICIP.2003.1246940) 8
- [27] A. Perer and B. Shneiderman. Balancing systematic and flexible exploration of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):693–700, 2006. doi: [10.1109/TVCG.2006.122](https://doi.org/10.1109/TVCG.2006.122) 2
- [28] D. Rappaport. A convex hull algorithm for discs, and applications. *Computational Geometry*, 1:171–187, 1991. doi: [10.1016/0925-7721\(92\)90015-K](https://doi.org/10.1016/0925-7721(92)90015-K) 8, 11
- [29] E. R. Tufte. *The visual display of quantitative information*. Graphics Press, second ed., 2001. 3
- [30] J. Vihrov, K. Prusis, K. Freivalds, P. Rucevskis, and V. Krebs. An inverse distance-based potential field function for overlapping point set visualization. In *Proc. 5th International Conference on Information Visualization Theory and Applications*, pp. 29–38, 2014. doi: [10.5220/0004681100290038](https://doi.org/10.5220/0004681100290038) 2, 3, 8, 9
- [31] J. Wagemans, J. H. Elder, M. Kubovy, S. E. Palmer, M. A. Peterson, M. Singh, and R. von der Heydt. A century of gestalt psychology in visual perception: I. perceptual grouping and figure–ground organization. *Psychological bulletin*, 138(6):1172, 2012. doi: [10.1037/a0029333](https://doi.org/10.1037/a0029333) 2, 3, 5
- [32] Y. Wang, D. Cheng, Z. Wang, J. Zhang, L. Zhou, G. He, and O. Deussen. F2-Bubbles: Faithful bubble set construction and flexible editing. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):422–432, 2022. doi: [10.1109/TVCG.2021.3114761](https://doi.org/10.1109/TVCG.2021.3114761) 3

# SimpleSets: Capturing Categorical Point Patterns with Simple Shapes

## Supplementary Material

Steven van den Broek, Wouter Meulemans, and Bettina Speckmann

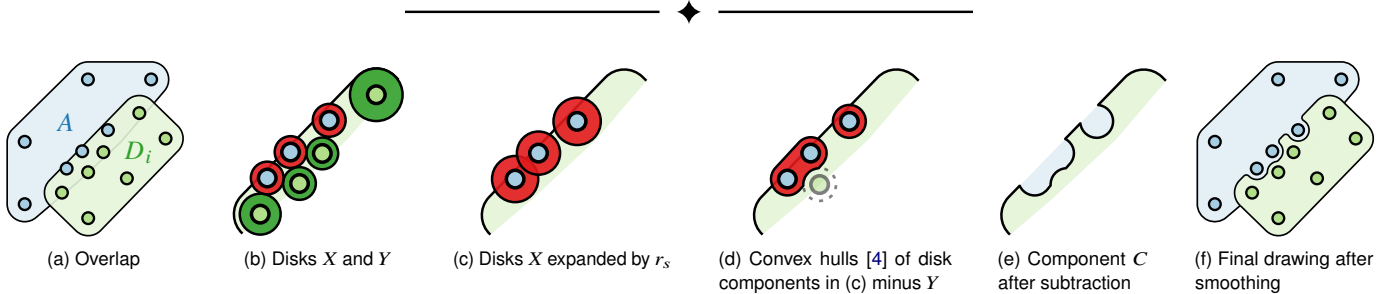


Fig. 1: Modification of dilated patterns to expose points beneath. Figures (b)–(e) show a closeup of the component  $C$  in (a) and (f).

### A EXTENDED DESCRIPTION OF THE CURVE MODIFICATION

**The general approach.** The modification process is summarized in Figure 1. Consider an arbitrary component  $C \in \mathcal{C}_i$  for some  $i$ . Let  $A$  be the set of patterns that are below  $D_i$  in the stacking order in the faces of  $C$ . We expose data points of patterns in  $A$  by modifying  $D_i$ .

Our algorithm aims to keep a disk of radius  $r_c = 5/8 \cdot r_d$  centered at each element visible. To this end, we place exclusion (red) disks of radius zero at each data point of each pattern in  $A$ . We also place inclusion (green) disks of radius zero at each data point of  $P_i$ . Then we grow the disks at a uniform rate. If two disks of different color collide then we stop their growth. We also stop the growth of a red disk when it reaches  $r_c$  and the growth of a green disk when it reaches  $r_d$ .

The result of this growth process is a set  $X$  of exclusion disks and a set  $Y$  of inclusion disks such that any disk in  $X$  is disjoint from those in  $Y$  and vice versa (Figure 1b). We now cut the disks in  $X$  out of  $D_i$  to expose the corresponding points and smooth the result using the Minkowski sum and difference with a disk of radius  $r_s = r_d/5$ . When the disks in  $X$  are sufficiently close (Figure 1c) then we cut out their convex hull [4] instead. We ensure that the disks in  $Y$  lie in the final pattern by not cutting them out (Figure 1d).

**Cutting out convex hulls.** We describe now more precisely when we group disks and cut out their convex hull. A disk in  $X$  is a candidate for a “group cut” only if the boundary of  $D_i$  it intersects is a line segment. Figure 2 illustrates the reasoning. A disk that does not intersect the boundary of  $D_i$  at all is also a candidate, see Figure 3 for details on this edge case. We expand these candidate disks by increasing their radius by the smoothing radius  $r_s$  (Figure 1c). We then determine the connected components of the intersection graph of the expanded disks. For each connected component, we compute the convex hull of its disks [4] and cut out any disk in  $Y$  (Figure 1d). The result is cut out of the part of  $D_i$  in component  $C$  (Figure 1e).

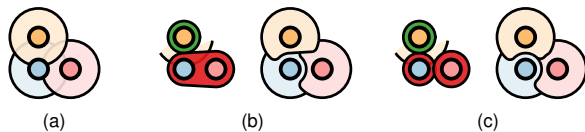


Fig. 2: We modify the top orange pattern to expose the bottom two points by cutting the disks in  $X$  out separately, even though they are close, because their intersection with the boundary of the top pattern is a circular arc. (a) Overlap. (b) Cutting out the convex hull of the two disks in  $X$ . (c) Cutting out the disks in  $X$  separately, our preferred solution.

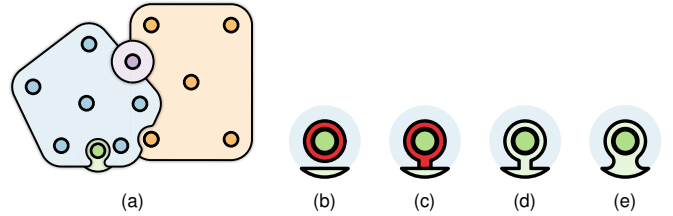


Fig. 3: (a) A setting where (b) a disk in  $X$  lies fully inside  $C$ . (c) We cut out a thin rectangle between the circle and the closest point on the boundary of  $C$ . In the end, (d) the modified component is smoothed, (e).

**Smoothing.** Let  $T$  be the boundary of  $D_i$  in component  $C$  after the cutting process;  $T$  is an open curve. We smooth by using the Minkowski sum ( $\oplus$ ) and difference ( $\ominus$ ) with a disk  $D$  of radius  $r_s = r_d/5$ . The shape of  $T$  beyond the cutouts should not be smoothed. To ensure that, we create a closed shape  $S$  by extending the endpoints of  $T$  in the direction of the respective tangents and close it around its bounding box (Figure 4a). The smoothed version  $S'$  of  $S$  is

$$S' = (((S \ominus D) \oplus D) \oplus D) \ominus D.$$

In terms of mathematical morphology operators [2], we first apply the opening operator to smooth convex vertices and then the closing operator to smooth reflex vertices. At the end, we cut  $S'$  to obtain the smoothed boundary  $T'$  (Figure 4b).

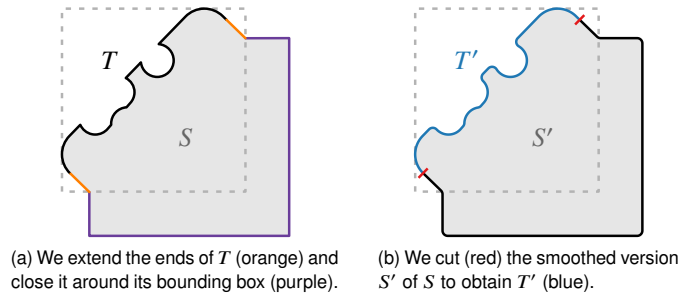


Fig. 4: Smoothing  $T$  into  $T'$ .

### B ADDITIONAL FIGURES

Figure 5 shows four SimpleSets partitions for different time parameters. Figure 6 illustrates that any set of disjoint patterns can be drawn with

the SimpleSets drawing algorithm. Figure 7 shows the equidistant point sampling used to measure inflections and curvature for the quantitative analysis. The remaining figures are outputs not present in the paper that were analyzed in the quantitative evaluation.

## REFERENCES

- [1] B. Alper, N. Henry Riche, G. A. Ramos, and M. Czerwinski. Design study of LineSets, a novel set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2259–2267, 2011. doi: 10.1109/TVCG.2011.186 2, 3, 12
- [2] R. M. Haralick, S. R. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):532–550, 1987. doi: 10.1109/TPAMI.1987.4767941 11
- [3] W. Meulemans, N. Henry Riche, B. Speckmann, B. Alper, and T. Dwyer. KelpFusion: A hybrid set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 19(11):1846–1858, 2013. doi: 10.1109/TVCG.2013.76 2, 3, 12
- [4] D. Rappaport. A convex hull algorithm for discs, and applications. *Computational Geometry*, 1:171–187, 1991. doi: 10.1016/0925-7721(92)90015-K 8, 11

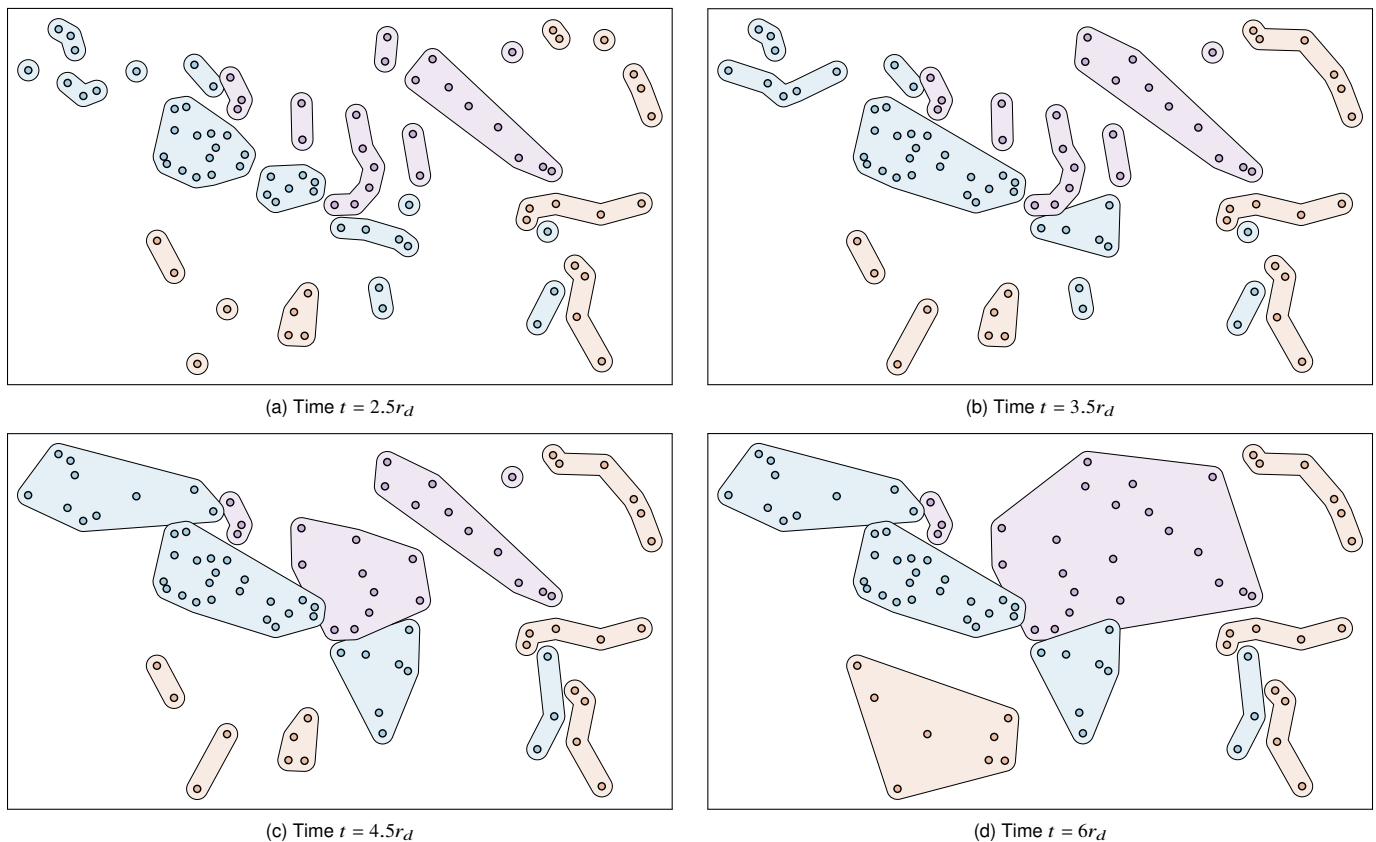
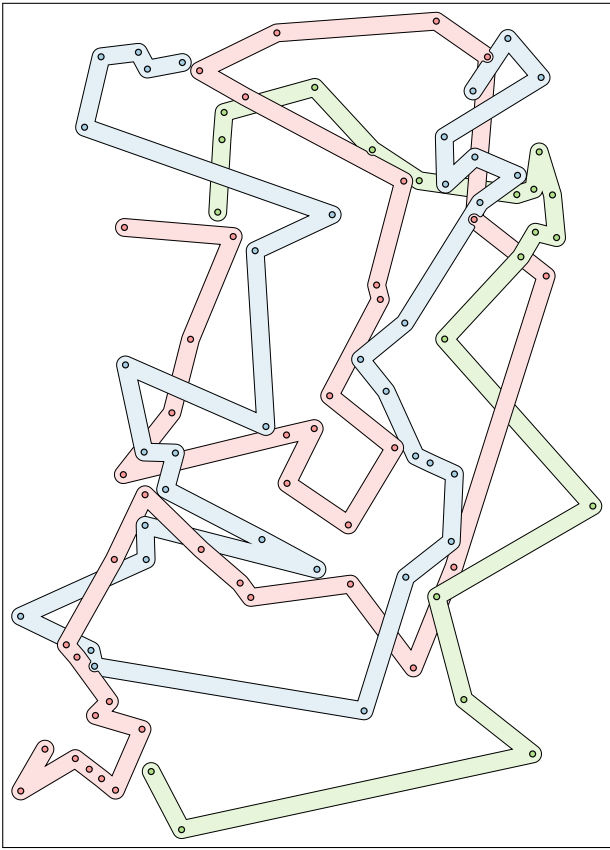
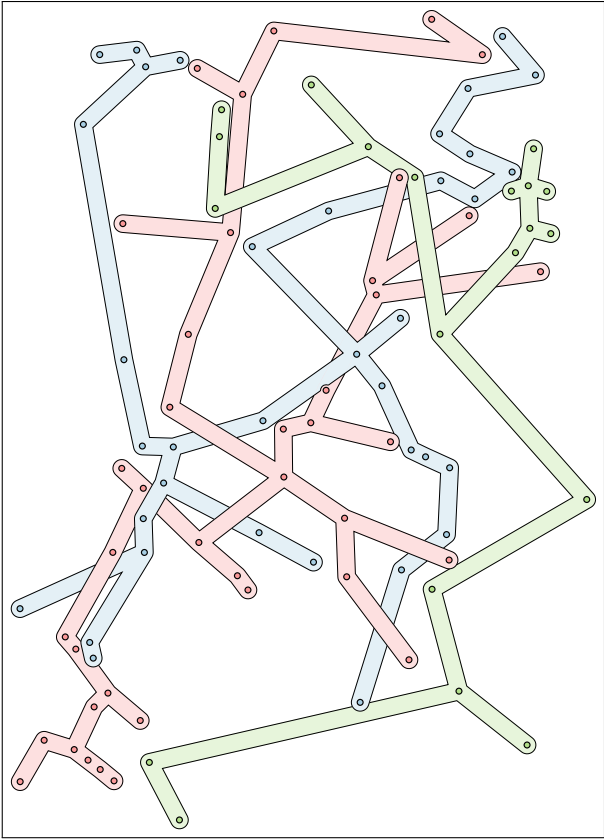


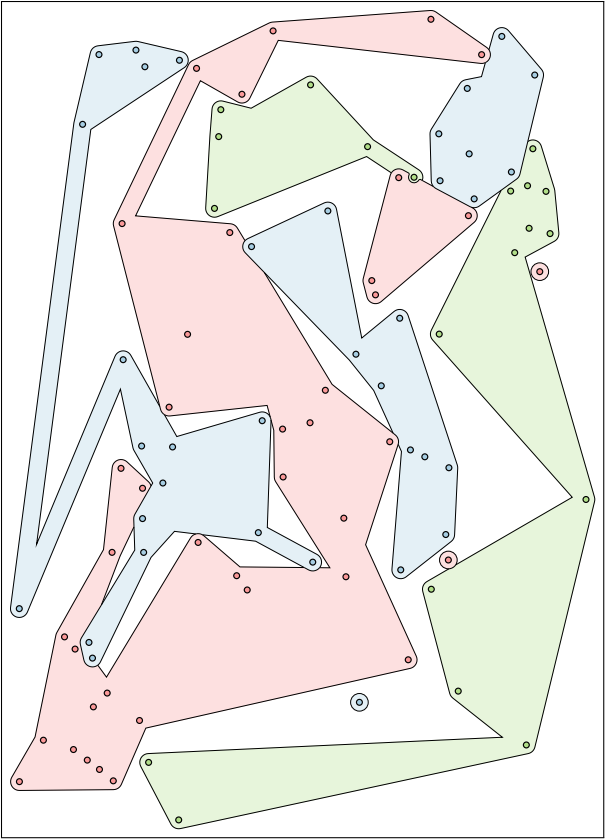
Fig. 5: SimpleSets visualizations of the Hotels dataset [1, 3] based on different partitions. Eleven points of the original dataset were removed because they belong to multiple sets.



(a) LineSets



(b) Bubble Sets



(c) ClusterSets

Fig. 6: Patterns from other visualizations drawn with the SimpleSets drawing algorithm.

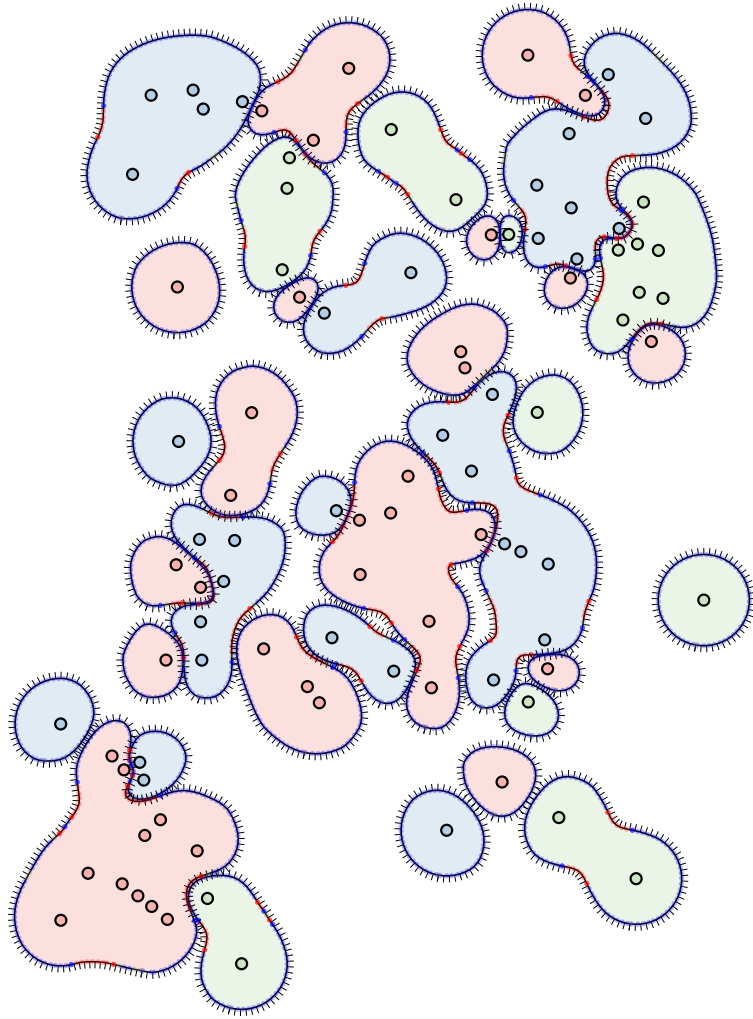
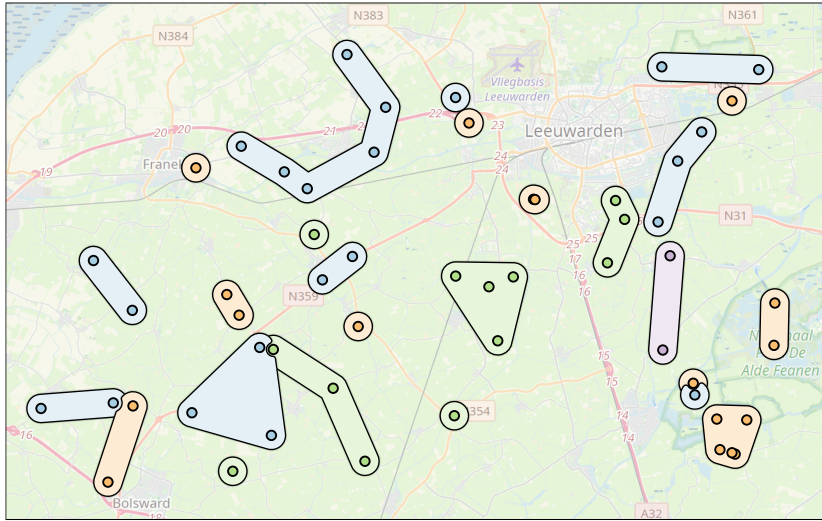
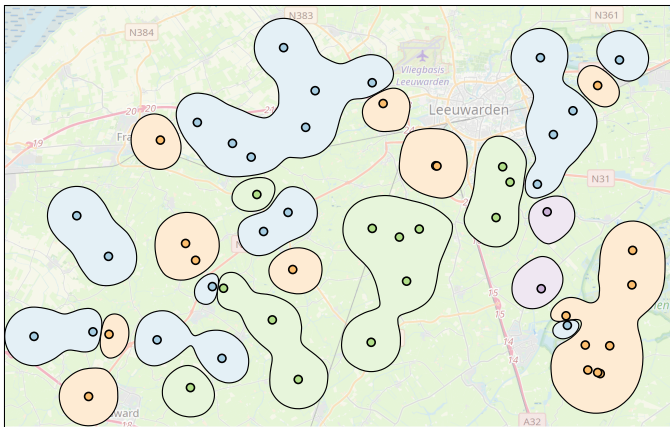


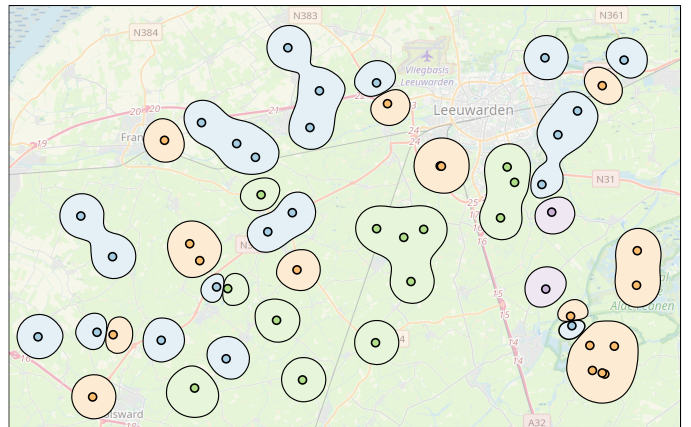
Fig. 7: Equidistant point sampling used to measure inflections and curvature. The figure shows VPF\*14+ on the NYC dataset. The black line segments show normals. The circles on the boundary show turns: clockwise (blue), counter-clockwise (red), and straight (grey).



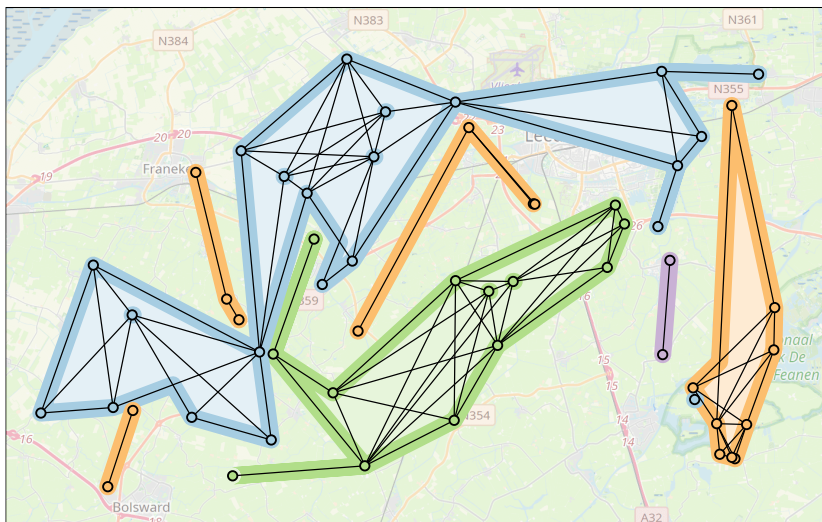
(a) SimpleSets



(b) VPF\*14+



(c) VPF\*14-



(d) ClusterSets

Fig. 8: Comparison on the Mills dataset. The ClusterSets and VPF\*14 outputs come from our implementation of their method.

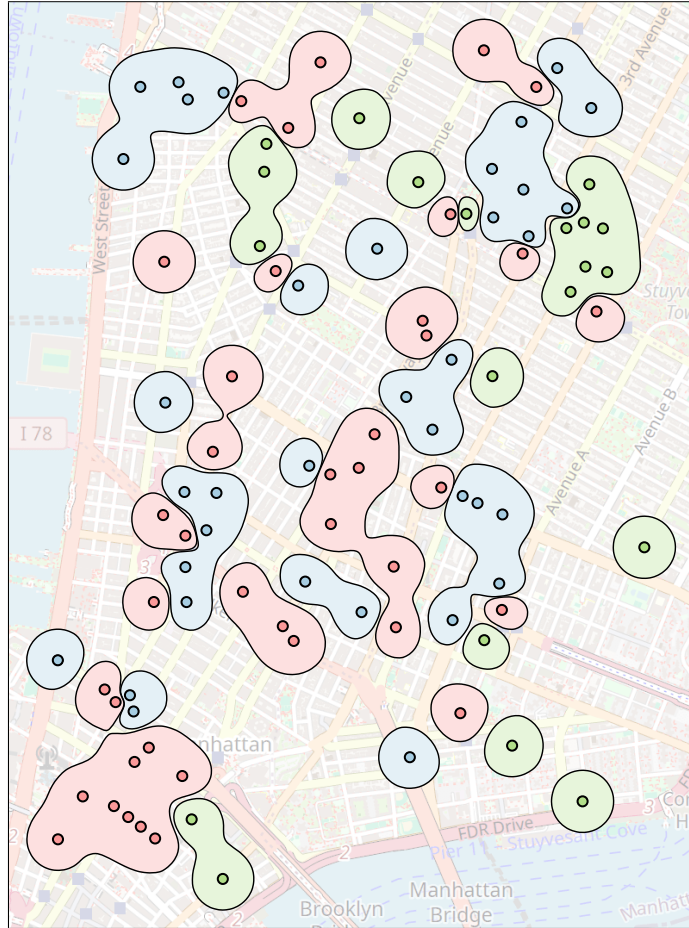


Fig. 9: VPF\*14 visualization on the NYC dataset with medium radius of influence.



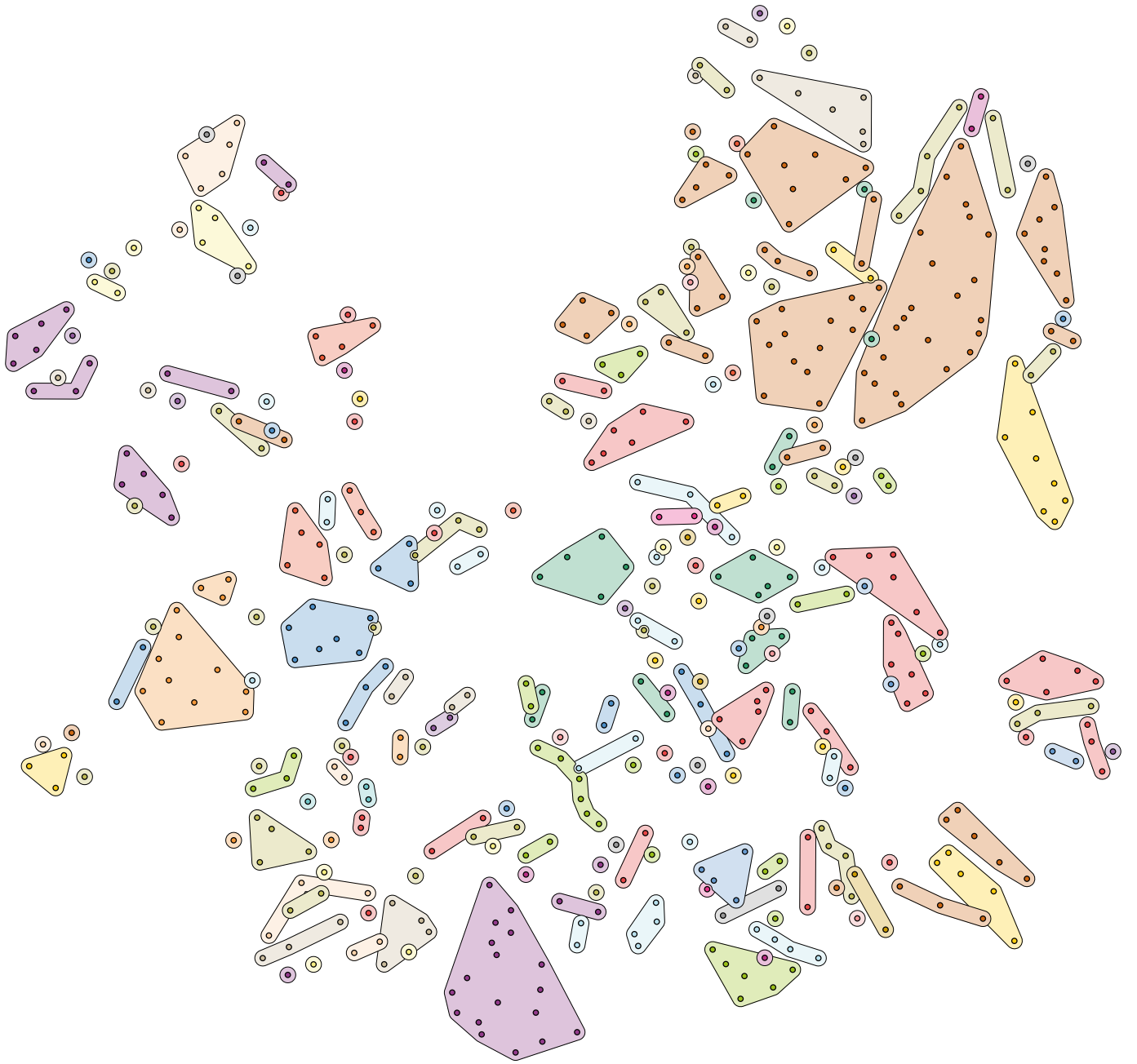


Fig. 10: SimpleSets visualization of the human disease network.

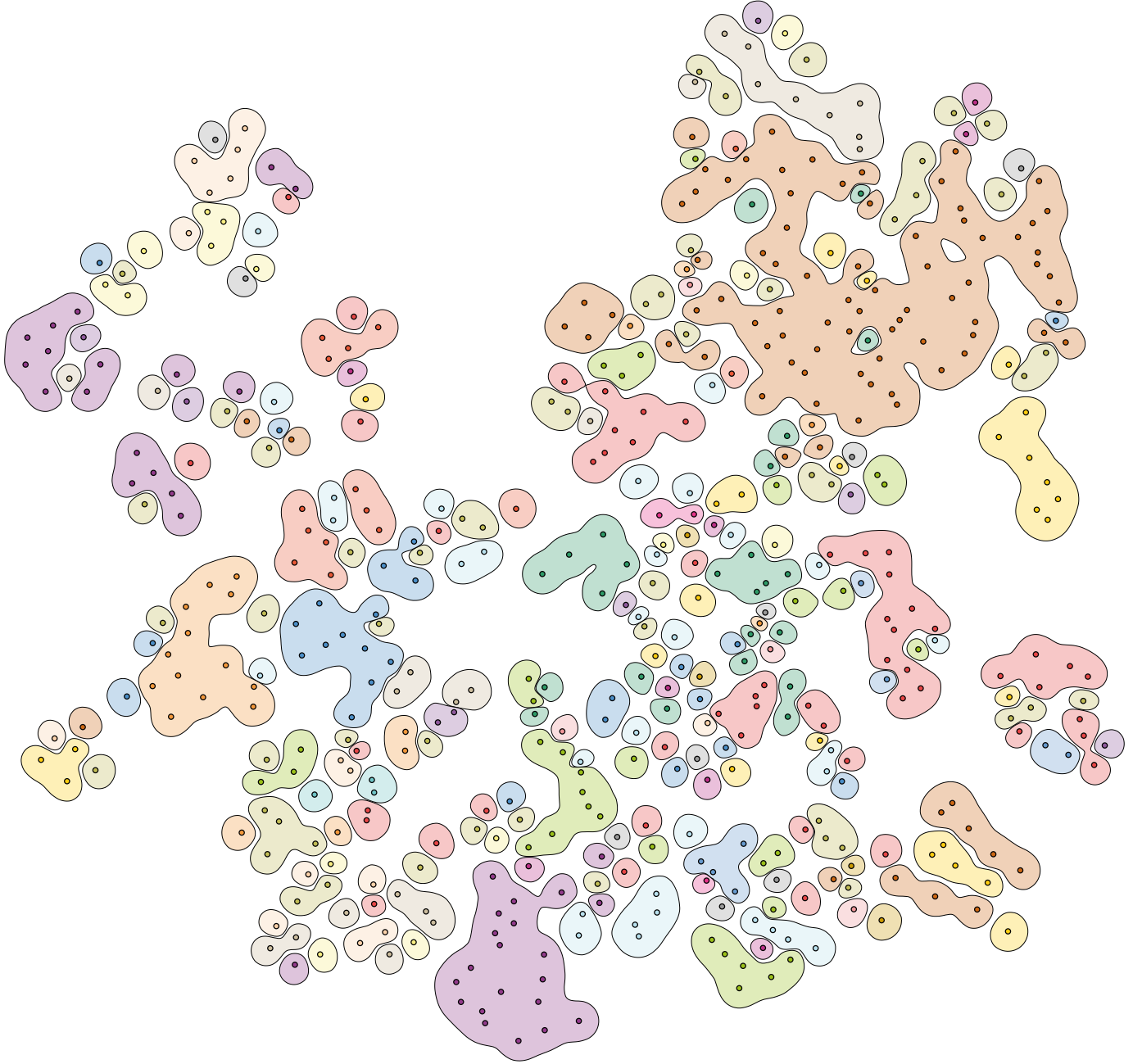


Fig. 11: VPF\*14 visualization of the human disease network; generated using our implementation of their method. Large radius of influence.

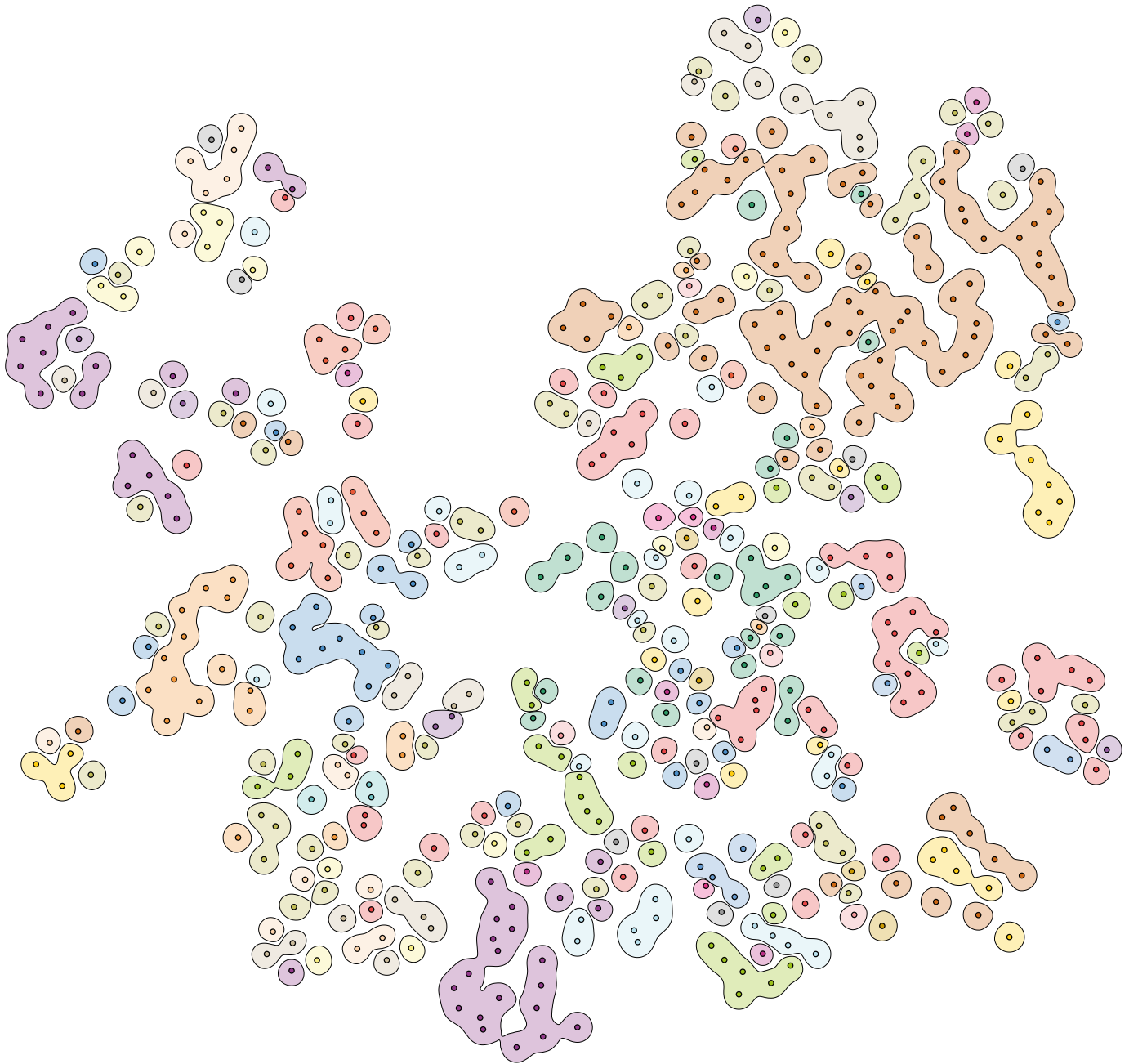


Fig. 12: VPF\*14 visualization of the human disease network; generated using our implementation of their method. Medium radius of influence.



Fig. 13: ClusterSets visualization of the human disease network; generated using our implementation of their method.