Edinburgh Research Explorer

A Generic Framework and Library for Exploration of Small Multiples through Interactive Piling

A Generic Framework and Library for Exploration of Small Multiples through Interactive Piling

Fritz Lekschas, Xinyi Zhou, Wei Chen, Nils Gehlenborg, Benjamin Bach, and Hanspeter Pfister
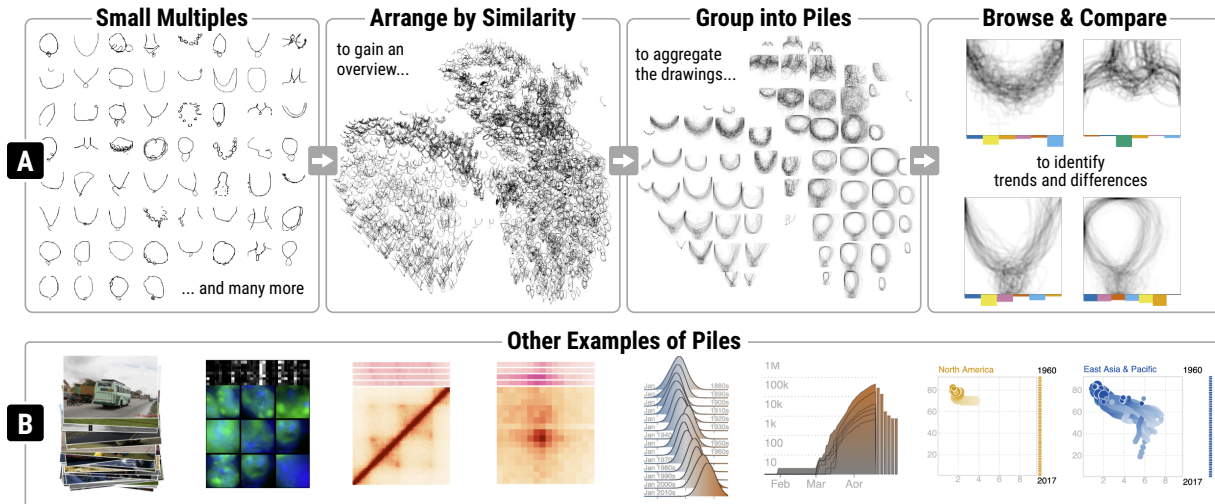
Fig. 1. **Exploring Small Multiples through Visual Piling.** (A) An example of thousands of necklace sketches from Google Quickdraw [25] displayed as small multiples. The interactive arrangement, grouping, and aggregation of small multiples into piles support the discovery and comparison of reoccurring patterns. (B) Other types of small multiple visualizations grouped and aggregated into piles, including (from left to right) natural and immunofluorescence microscopy images, matrices, area charts, and scatterplots.

Abstract— Small multiples are miniature representations of visual information used generically across many domains. Handling large numbers of small multiples imposes challenges on many analytic tasks like inspection, comparison, navigation, or annotation. To address these challenges, we developed a framework and implemented a library called PILING.JS for designing interactive *piling* interfaces. Based on the piling metaphor, such interfaces afford flexible organization, exploration, and comparison of large numbers of small multiples by interactively aggregating visual objects into piles. Based on a systematic analysis of previous work, we present a structured design space to guide the design of visual piling interfaces. To enable designers to efficiently build their own visual piling interfaces, PILING.JS provides a declarative interface to avoid having to write low-level code and implements common aspects of the design space. An accompanying GUI additionally supports the dynamic configuration of the piling interface. We demonstrate the expressiveness of PILING.JS with examples from machine learning, immunofluorescence microscopy, genomics, and public health.

Index Terms—Information visualization, small multiples, interactive piling, visual aggregation, spatial organization.

✦

1 INTRODUCTION

In many disciplines, datasets consist of large numbers of elements, pattern instances, or dimensions. For instance, in supervised machine learning, researchers compile sets of photos to train and validate machine learning models; in genomics, computational biologists study visual patterns that act as proxies for biological features; in public health, medical experts try to correlate different measurements to health conditions of their patient cohort.

Small multiples [46] are a widely used visualization technique to display such datasets through a series of miniature visualizations that show different facets or subsets of the data. However, as the number

of small multiples grows, comparison and exploration can become inefficient due to the decreasing availability of screen real estate per visualization and the increasing efforts for sequential scanning. Subsampling or filtering can help to limit the number of small multiples but might obscure important characteristics of the dataset. Summary visualizations can alleviate this problem by aggregating subsets of the data into a single visualization. However, the analyst needs to know upfront how to organize the dataset into subsets. Without interactive features, exploration with summary visualizations can be limited when there are many potentially interesting facets or subsets to explore.

We propose a generic framework for exploring large numbers of small multiples through interactive visual *piling*. Inspired by how physical piles enable casual organization [35] of paper documents, piling in visualization affords spatial grouping of visual elements into piles that can be arranged, browsed, and aggregated interactively. By combining the benefits of small multiples and visual aggregations with interactive browsing, piling can be an effective technique for exploring small multiples. For instance, in Fig. 1A, we demonstrate how piling enables the discovery and comparison of shared concepts of necklace sketches through interactive arrangements, groupings, aggregation, and browsing. Currently, piling has been applied to matrix visualizations by ad-hoc domain-specific methods to explore set typed data [43], dynamic

networks [3], and matrix patterns [31]. But there are many more scenarios where interactive visual piling can be useful for exploration, as shown in Fig. 1. However, developing a new piling system for every use case would be time-consuming. Moreover, many concepts of piling are independent of the domain, data type, and visual encoding.

We present the first overview of interactive visual piling as a method for the exploration of small multiples. Based on a systematic analysis of previous work, we define a design space according to five analytical tasks that any piling interface should support: grouping, arrangement, previewing, browsing, and aggregation. We focus on analytical tasks to study how different visual encoding and interaction approaches support the exploration of small multiples. Our design space provides guidance for the design of future piling interfaces. To streamline the implementation of piling interfaces, we developed PILING.JS—a JavaScript-based library that provides solutions for many common aspects of our design space. PILING.JS is built around a data-type independent rendering pipeline and a declarative view specification to avoid having to write low-level code for handling the interactive piling interface. Using PILING.JS, we demonstrate the generality of interactive piling for exploring small multiples and the expressiveness of our library with examples from machine learning, immunofluorescence microscopy, geography, public health, genomics, and more. The source code of PILING.JS is freely available under a permissive open source license at https://github.com/flekschas/piling.js and features extensive documentation and examples at https://piling.js.org.

2 RELATED WORK

Small Multiples. Small multiples [46] are a series of miniature visualizations that show different facets or subsets of a dataset or different instances of a pattern type. Small multiples afford direct visual comparison and are, for example, used for multifaceted exploration [7, 21], analyzing temporal data [10, 40], parameterization [24, 36], or as a general exploration method for visual analytics [47]. Small multiple designs are conceptually similar to glyph design [41] as the reduced availability of screen real estate render glyphs useful. Typically, small multiples are positioned in a grid or data-driven layout [44, 49] that the user cannot manipulate directly. With visual piling, the goal is to enhance small multiples to support interactive grouping and aggregation of large numbers of small multiples for scenarios that require comparisons across a multitude of facets and subsets of the data.

Piling for Document Organization. Piling is a technique for the spatiotemporal organization of documents. Based on studying physical piling, Thomas Malone [34] suggests that piling is cognitively easier than filing as it only involves loose classification of documents. In a study conducted by Whittaker and Hirschberg [51], physical piling resulted in more frequently-browsed data collections compared to a folder-based exploration approach. Mander et al. [35] introduced piling as a technique for casual organization of documents in a virtual desktop environment. To address the scalability issues of piling, they experimented with automatic piling strategies and different modes of visually encoding and interacting with piles to aid document retrieval and browsing. In follow-up work [42], they show how automatic piling-based grouping and aggregation can enhance content-aware browsing of virtual document collections. Kim et al. [28] employed visual piling for browsing photos and found automatic piling to be as efficient for search as manually sorting and more efficient compared to automatically ordering. In this work, we are expanding the piling approach for casual document organization into a generic technique for interactive visual aggregation.

Interacting with Visual Piles. In the human computer interaction community, several projects explored interaction techniques for interactive browsing of document-based piles. For instance, using a 3D virtual desktop environment called BumpTop [1], Agarawala et al. studied and implemented several pen-based interaction techniques for a tabletop display. BumpTop explores leafing through items like one would leaf through pages of a book, partially dispersing piles to see individual items better, or temporarily dispersing piles into a grid of items to avoid any overlap. Aliakseyeu et al. [2] have further studied pen-based interaction techniques for browsing piles and found dispersing to be

most effective. Other work explored the space of tangible interactions with piles in a mixed-reality environment such as digital tabletops [27] or bendable e-ink displays [19].

Additionally, Bauer et al. [5, 6] have explored spatial arrangement techniques for piling in Dynapad, where a pile is more loosely defined as a spatially-constrained set of visual items. Items must not necessarily overlap, which allows for continual exposure of items but requires more space and does not support aggregation. Another interaction technique for spatially organized items, called Bubble Clusters [50], is built on the implicit formation of groups based on their spatial proximity. WallTop [8] implements a similar approach, where overlapping windows feature an outline that allows for group-based spatial positioning via drag-and-drop. We analyzed this work to identify common gestures for interaction, which we implemented in PILING.JS.

Piling for Information Visualizations. In information visualization, visual piling is used for comparison. Tominski et al. [45] developed a generic interactive technique for pairwise comparison of information visualizations inspired by how people compare physical sheets of paper. Their technique is similar to piling for pairwise comparisons, but it does not visually or interactively scale to more than two items. Beyond this work, piling has mainly been applied to matrices for visual aggregation. For example, the Onset [43] technique implements a piling interface to interactively aggregate binary matrices for comparison. Bach et al. extended this approach to dynamic networks in Small Multipiles [3] for detecting states over time. Importantly, Bach et al. introduced the notion of a *preview* representation for items, which they implement as one-dimensional aggregates of the 2D matrix to aid browsing. In HiPiler [31], the idea of piling is further generalized to support one-, two-, and multi-dimensional arrangements. Vogogias et al. [48] and Fernandez et al. [17] have applied similar matrix-based visual piling ideas to other applications in biology and software evolution. Finally, Lekschas et al. [30] use piling to guide navigation in multiscale visualizations by aggregating overlapping patterns into piles and displaying them as insets.

In general, visual piling is an approach to reduce clutter [15] through interactive aggregation. In this paper we demonstrate the usefulness of piling and generalize the piling approach beyond matrix visualizations.

3 THE VISUAL PILING APPROACH

Visual piling is an interactive approach for organizing, exploring, and comparing small multiples. Piling is centered around the act of spatially positioning items on top of each other, which together form a pile, and arranging these piles meaningfully to support effective comparison.

3.1 Elements and Properties of a Visual Pile

Inspired by physical piles of paper documents, we define a pile as a group of partially-occluded small multiples that results from piling up individual *items*, illustrated in Fig. 2. Given the partial overlap, only a single item is shown in its entirety, which we call the pile *cover*. As the remaining items are only partially visible, we refer to them as *previews*. While there are many ways of visually representing a pile, we distinguish piles from other forms of spatially-arranged small multiples by the following set of properties, which builds upon the description of a physical pile from Bauer et al. [6].

Occlusion & Connectedness. Items that comprise a pile should occlude each other partially to form a single mutually-connected unit. However, piles can temporarily be dispersed for exploration.

Identity. A pile must differentiate itself visually from a single item. There are different visual encodings to identify a pile, like a label indicating the number of items on a pile, superimposed semi-transparent images, or items that are offset relative to each other.

Cohesion. Items on a pile should act as a single element during the exploration. Cohesive behavior does not mean that access to individual items is lost upon grouping items into a pile. However, a pile should reflect the notion of a group when interacting with the piling interface.

Transience. Piling should be seen as a dynamic process where the piling state can change frequently. In contrast to other aggregation techniques, the goal of piling is to compose and disperse piles interactively
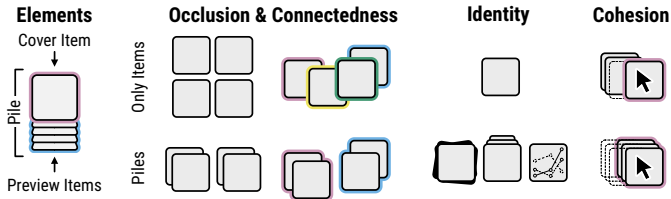
Fig. 2. **Elements and Properties of Visual Piles.** To illustrate key properties of piles, we differentiate between individual items and piles.

rather than to just consume a static grouping state. However, this does not mean that piles cannot persist.

3.2 Goals and Tasks

Even though the application-specific goals differ, we identify two overarching goals for interactive visual piling interfaces from related work. (**G1**) Visual piling is a tool for organizing data collections into subsets to reduce complexity. This includes, for example, to sort items into groups, categorize groups based on their content, or filter out subsets of items for quality control. (**G2**) Beyond organization, visual piles are a means to explore and compare individual items and groups of items to each other. Specifically, one might want to determine the primary topic of a group, identify outliers, or discover trends.

To identify the common tasks needed to support organization, exploration, and comparison, we systematically reviewed related work. Following an open-coding approach, the first two authors coded all 17 piling-related papers from Sect. 2 according to their application-specific tasks independently. We focused our coding efforts on the role of interactive piling to not confuse piling-specific with unrelated tasks. After coding the papers, the first two authors resolved disagreements. Subsequently, we generalized the assigned codes into five high-level analytic tasks that any interactive visual piling interface should support.

T1 **Grouping:** *manually* or *automatically* sort items into piles.

T2 **Arrangement:** position items and piles relative to each other in an *orderly*, *randomized*, *gridded*, or *unconstrained* layout.

T3 **Previewing:** identify and locate items on a pile using *in-place*, *gallery*, *foreshortened*, *combining*, and *indicating* previews.

T4 **Browsing:** search, explore, and navigate within and between piles through *in-place*, *dispersive*, *layered*, and *hierarchical* browsing.

T5 **Aggregation:** summarize a pile into a *synthesized*, *representative*, or *abstract* representation.

To study how different visual encoding and interaction approaches support the exploration of small multiples, we use these five analytical tasks to structure the design space exploration (Sect. 4) and to guide future piling designs.

3.3 Usage Scenario

To exemplify how visual piling enhances the exploration of small multiples, we describe a typical usage scenario following the example of necklace sketches from Google Quickdraw [25] (Fig. 1A), which we also demonstrate in the supplementary video. One goal in analyzing large collections of visual objects is to identify and compare trends within the dataset. Inspired by Forma Fluens [37], we are trying to find reoccurring pattern concepts. Visualizing the sketches as small multiples (Fig. 1A1 left) allows us to assess and compare individual sketches, but it does not support the discovery of shared concepts. A common approach to uncover similarities within large and high-dimensional data collections is to arrange (**T2**) the items as a two-dimensional embedding for cluster analysis (Fig. 1A2). We arranged the items by a UMAP [38] embedding of image features that were learned with a convolutional autoencoder. In the resulting cluster plot, items can be represented as a symbol (e.g., a dot) or a small thumbnail. While symbol-based cluster plots are highly scalable, they do not reveal the visual details of a cluster. On the other hand, thumbnail-based cluster plots do not scale to large datasets due to overplotting issues. Visual piling provides a trade-off by grouping (**T1**) spatial clusters, i.e., clusters formed by items in relative proximity (Fig. 1A3). By aggregating (**T5**) all sketches into an average and showing this average as the pile cover, we can discover and *browse* overarching concepts effectively. For instance, after manually refining the grouping and arrangement of four piles (Supplementary Figure S1), we can see that people are sketching a necklace as an open beaded necklace, a necklace worn around a neck, an open pendant necklace, or a closed pendant necklace (Fig. 1A4). Visual piling also affords the encoding of additional information beyond the individual items. For instance, in Fig. 1A4, we visualize the relative distribution of geographic regions across a pile using small bar charts below each pile.

4 A DESIGN SPACE FOR VISUAL PILING

This is the first design space (Fig. 3) for visual piling. For each of the five analytical tasks (Sect. 3.2), we derived general approaches and common solutions from previous work through multiple discussions among the co-authors. The resulting subcategories cover overarching approaches of each task. We generalize these approaches to highlight conceptual differences. Multiple approaches can be combined to offer different ways of organizing and exploring small multiples. In our design space, we cover the relevant visual encodings and interactions. We also describe common gestures for triggering interactions but do not attempt to provide a complete overview of all possible gestures.

4.1 Grouping

We distinguish between *manual* and *automatic* grouping (**T1**), as exemplified in Fig. 3 Grouping. Manual grouping requires the user to interactively determine which items should be grouped and, potentially, in which order. Automatic grouping follows a specific procedure to group multiple items at once.

Manual. *Sequential* grouping is the simplest form of manual grouping. It requires the user to group items interactively, one at a time. This is typically achieved with a drag-and-drop gesture [1,3,5,31,35,43,50]. While sequential grouping requires more time, it enables temporal organization. For instance, the most recently added elements can be located on top of the pile. For efficiency, one can also form a group from multiple selected items. While multi-select grouping does not result in intermediate groupings, the sequence of selected items can still be reflected, given the order of selected items. In contrast, *parallel* grouping techniques allow two or more items to be piled up at the same time. For instance, many piling interfaces support region-based grouping via lasso techniques [1,31]. Parallel grouping does not afford temporal organization as the order in which multiple items are grouped together is not explicitly defined. A special form of grouping, which can be treated as parallel or sequential, is swiping [31], where the user moves the mouse cursor or pen over each item to be grouped. Swiping enables more precise selections in dense arrangements like cluster plots.

Automatic. Many piling interfaces support automatic grouping to improve scalability. *Layout*-driven grouping is based on an explicitly- or implicitly-defined layout. Items that are located within the same unit of the layout can then be grouped. Such units can, for instance, be the rows, columns, or grid cells [31]. *Proximity*-based grouping uses the Gestalt principle of "proximity," which states that nearby items implicitly form groups perceptually. However, implicit grouping can cause uncertainty in subsequent pile interactions [35] as it is not always possible to infer the grouping state as perceived by the user [6]. Therefore, most piling interfaces only use proximity to trigger explicit grouping, e.g., by outlining the pile bounds [5,50] or merging nearby items [30]. Finally, in *similarity*-based grouping, items are merged automatically based on some notion of similarity. While there are many different ways of measuring similarity, fundamentally, the similarity can be derived from the items [3,28,31,42] or related metadata [31].

4.2 Arrangement

For arrangements (**T2**), we consider the relative positioning of items on a pile and the absolute positioning of piles (Fig. 3 Arrangement).
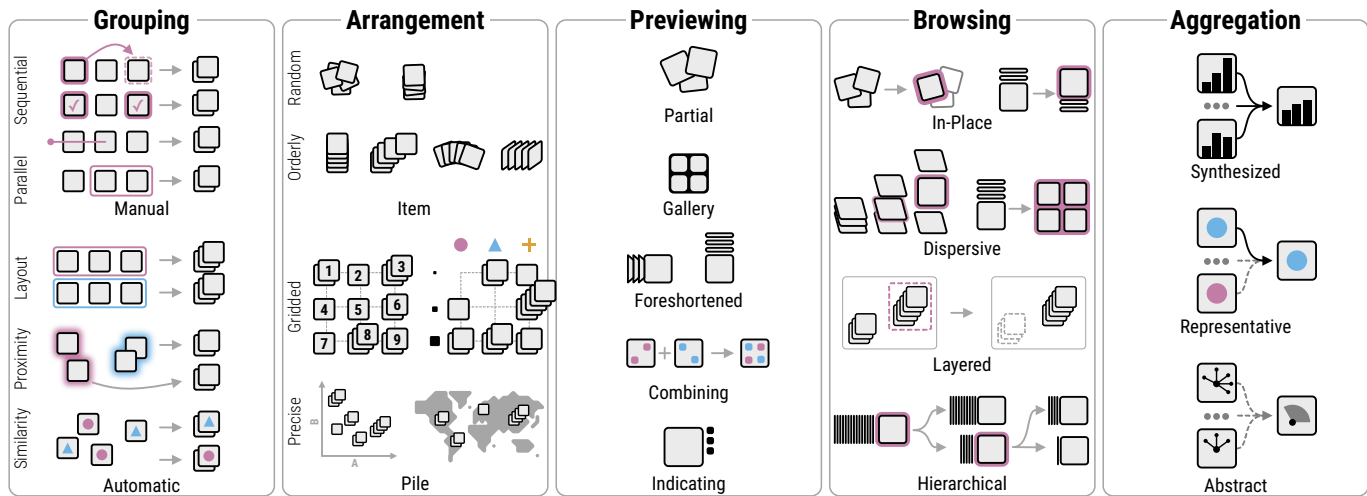
Fig. 3. **Dimensions of the Visual Piling Design Space.** We structure the design space according to the five analytical tasks into different approaches for grouping, arrangement, previewing, browsing, and aggregation.

Item arrangement. A *random* item arrangement is characterized by non-deterministic offsets and rotations of the items. Such arrangements make the visual pile closely resemble a physical pile, which can be useful to distinguish between automatically- and manually-composed piles [35]. Random item arrangements can also encode access patterns, such as the frequency of file access [14]. Finally, a pseudo-random item arrangement can be the result of sequential grouping. Since it is unlikely that the user will stack items in a pixel-precise manner, the resulting offset can appear random. Nevertheless, the offset can provide meaningful cues to the pile creator [34] for browsing (**T4**). In contrast, *orderly* item arrangements are the result of automatic and deterministic positioning. Such arrangements enable controlling how much each item is overlapped, which is useful for comparing items [3, 31]. When the item offset follows a single direction, *in-place* browsing can be efficient as the cursor movement is minimal. Also, orderly arrangements typically follow the Gestalt principle of "continuation" to foster perceptual grouping.

Pile Arrangement. Dividing the canvas into rows and columns of a specific size leads to a *gridded* pile arrangement. Gridded arrangements are useful for comparing piles due to the alignment. Imposing a specific ordering onto the pile can highlight temporal or sequential patterns. The simplest form of a gridded pile arrangement is a one-dimensional timeline [6, 28] but two-dimensional grid layouts are more common to make use of the entire screen [1, 3, 5, 31]. Finally, as *precise* pile arrangements, we summarize manual, layout-driven, or data-driven arrangements that require a pixel-precise positioning on the canvas. In this regards, automatic arrangements can incorporate one-dimensional [6], two-dimensional [31], or multidimensional [31] scatterplots. The position can also be inherent to the items themselves, which is, for example, the case for exploring annotated pattern instances [30].

4.3 Previewing

To afford content-awareness, visual piles can implement different layout types to support previewing items (**T3**), as shown in Fig. 3, which is key to support effective exploration and navigation (**T4**).

Partial. Inspired by physical piling, partial previewing of items arises naturally and is implemented in many piling interfaces [1, 2, 5, 27, 28]. The effectiveness depends on the data type and the size of the partial previews.

Gallery. When the partial overlap severely limits the perception of the item's content, one can opt for a gallery preview where a small number of items or aggregates [30] is shown in a regular grid. This approach can be useful for datasets in combination with a representative aggregation approach (Sect. 4.5).

Foreshortened. To limit the size of a pile while still providing item-specific previews, previews can be foreshortened along one axis.

Such previews can be implemented with perspective distortion [1], compression, or aggregation along one dimension [3, 17, 30, 31, 48]. While the first option can provide cues for search and navigation (**T4**), the latter enables more effective comparison between alignable items.

Combining. When working with items that have a sparse visual representation and shared axes, like scatterplots, line charts, or bar charts, multiple items can be combined to provide an overview. A combined preview can be the result of superimposing items with a transparent background or through dedicated aggregation [43]. While this approach is space-efficient, the relationship between the overview and individual items might get lost without employing other means of previewing items.

Indicating. To maximize space efficiency while still hinting at an item's content, one can provide abstract indicators as item previews. Most common indicators are implemented as tabs [23, 43], but the indicator can be more abstract, e.g., a small dot. While indicators do not directly preview the content, they afford browsing (Sect. 4.4) and can encode metadata like the distribution of items.

4.4 Browsing

In the context of visual piling, we regard browsing (**T4**) as the act of inspecting the visual details of piled items (Fig. 3 Browsing). Since browsing requires interaction, the applicability of different browsing approaches depends on the type of preview.

In-Place. Inspecting the visual details of an item in-place is a fast browsing approach as the arrangement of items remains unchanged. To show an item in its entirety, the ordering of items is altered temporarily such that the browsed item is shown on top [1, 2, 23]. A variation of in-place browsing shows the browsed item next to the pile as an inset [35]. Another in-place browsing technique called "leafing" [1, 3, 30, 31] involves interacting with foreshortened previews. Upon interaction, foreshortened previews can either be expanded to their full extent or shown on top of the pile, similar to flipping through the pages of a book. Typically, in-place browsing is triggered by moving the pointing device over the preview item to be shown in its full extent [1–3, 23, 30, 31, 35].

Dispersive. Dispersive pile browsing techniques temporarily disperse a pile such that the overlap between items is resolved partially or entirely, allowing for subsequent comparison of the items. To aid maintaining a mental map of the items on a pile, many dispersive techniques use the same type of layout for positioning the dispersed items and only increase the spacing between items [1, 3, 50]. A more disruptive approach arranges the dispersed items into a regular grid [1, 2, 6]. Pile dispersion is often triggered by a double click or tap [2, 3, 31, 50], but many gestures have been explored too including horizontally moving a pointer device back and forth [35], hovering over a pile [1], and context-menu induced dragging gestures [1]. Finally, an indirect way
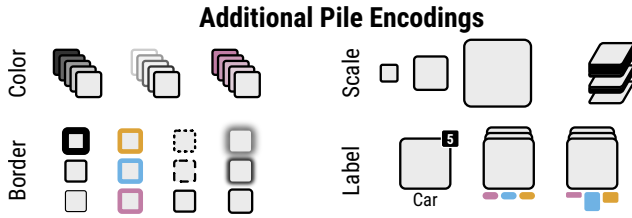
Additional Pile Encodings

Fig. 4. **Additional Pile Encoding.** The coloring, scale, border, and labels afford additional encodings that can be useful for exploration.

of dispersive browsing employs a zoom gesture in combination with automatic proximity-based grouping [5, 30]. Thereby, piles gradually disperse as the user zooms into a specific region.

Layered. Increasing numbers of small multiples limit the available space for visual browsing. Layered browsing techniques temporarily hide other items and, thus, give more space to the browsed piles. Layering can be combined with dispersive browsing to support flexible pile exploration and sub-piling [31].

Hierarchical. Finally, for piles of many items, it can be ineffective to browse all items at once. Instead, hierarchical clustering can be employed to enable hierarchical browsing such that the pile only disperses into a subset of piles from the next hierarchical level.

4.5 Aggregation

Aggregation (**T5**) is the act of summarizing piled items into a concise form (Fig. 3 Aggregation). The goal of aggregation is to improve the content awareness when previewing a pile and aid comparison between groups of items. Therefore, the choice of the aggregation method depends on the layout type for previewing.

Synthesized. We call aggregation techniques that create a single image from a group of items synthesized aggregations. Hereby, the resolution or granularity of the aggregate is identical to the items. Summary statistics are commonly used for synthesized aggregations [3, 30, 31, 43, 48] but other methods are possible. When the aggregate presents new or unseen information, it is useful to provide some means of previewing individual items [3, 31, 43] to enable item-specific comparisons.

Representative. For data types where individual items do not align well, such as natural images, synthesized aggregations are typically ineffective. Instead, the pile can be summarized by a single or multiple representative items which are typically visualized as gallery previews [30]. Through careful sampling, the selection of representative items can provide enough information to inform the user about a pile's primary content.

Abstract. Finally, for non-alignable but well-defined data, one can employ abstract aggregation techniques. The goal of such techniques is to provide a simplistic or schematic representation of the pile's content where the resolution or granularity is reduced compared to the items. Simplistic aggregations provide limited content awareness. However, the aggregation can, nevertheless, hint at the category or type of items on a pile, which can be useful for navigation (**T4**).

4.6 Additional Pile Encodings

Several additional style properties (Fig. 4) can be employed to encode related information such as the pile or item size, item access patterns, or categorical information.

Coloring. To highlight trends within a group of piled items, one can adjust the lightness [35] or apply other color filters. While potentially effective at encoding additional information, extreme color adjustments can harm content awareness and should be used with caution.

Scaling. Scaling the visual pile size in the x,y [30] or z-direction is another approach to encode additional information. Z-scaling requires that items are represented as three-dimensional objects. If applied, z-scaling can also afford edge browsing, which is an in-place browsing technique for physical piles [35].

Border. Beyond manipulating the piled items, visual piles can also utilize a border to encode information through its color, size, texture, or sharpness [30, 31].

Labeling. When working with categorical data, grouping and comparison can be improved by drawing a textual label [3, 14, 35] or visualizing categories as colored badges [31]. These badges can be adjusted to show the distribution of categories across a pile for improved browsing and comparison, as shown in Fig. 1A "Browse & Compare".

5 PILING.JS– A LIBRARY FOR VISUAL PILING

Based on our design space definition, we developed PILING.JS, a JavaScript library, to streamline the implementation of piling interfaces. PILING.JS is built around a data-agnostic rendering pipeline and a declarative view specification to define complex piling interface without having to write low-level code (Supplementary Figure S2). The library manages the view state and implements methods for interactive grouping, arrangement, browsing, and previewing. The designer's task is to specify the data rendering and aggregation (i.e., the elements of a visual pile), and the visual pile encoding. For interaction, gestures to manipulate piles are provided automatically by PILING.JS. See Supplementary Table S1 for an overview of PILING.JS' coverage of the visual piling design space.

5.1 Data Rendering

As a first step, the designer needs to specify the data rendering pipeline (Fig. 5) to create the elements of a visual pile. This pipeline involves the definition of the item data, a renderer, and an aggregator. While all three aspects play together, their implementations are decoupled so that each component can be reused, replaced, and extended easily.
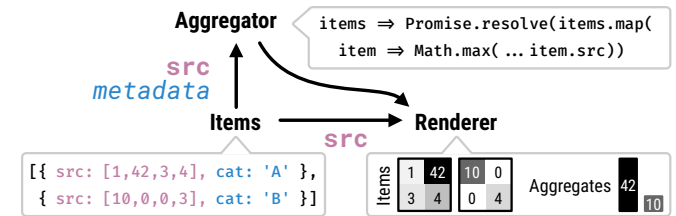

Fig. 5. **Rendering Pipeline.** Renderers receive as input the `src` property of `items` or the result of an aggregator. The aggregator in turn receives the `items` as input including additional properties (blue).

Data. PILING.JS operates on two primary data concepts: `items` and `piles`. The `items` are determined by the user and treated as immutable objects, while `piles` are created dynamically by PILING.JS. Each item is expected to be a JavaScript object with a `src` property that is, by convention, passed to the renderer (Fig. 5). This convention allows the designer to switch between different renderers without having to change the item itself. To enable exploration of dynamic datasets, PILING.JS supports a data matching strategy, using an optional `id` property, which works similarly to D3's [9] data joins. Upon updating the `items`, PILING.JS will match items by their `id` to identify outdated, updated, and newly-added items. By default, the `id` is set to the list index of `items`. An item object can contain additional properties to be used for dynamic styling, which enables concise and readable style specifications (Fig. 8). The `piles` are a list of objects that hold the item `ids` and the pile's xy position. All other pile encoding aspects are inferred automatically from the view specification. We chose this relatively simple data model to make piling more accessible.

Renderers. PILING.JS renders items into textures and subsequently operates on these textures during gesturing to decouple the state management and gesture handling from the domain-specific visualizations. To support a wide range of previewing and aggregation approaches, PILING.JS offers different rendering regimes for items, previews, and covers (Fig. 6). A renderer is a function that receives as input the `src` property of one or more items and returns a single *promise* object that resolves to texture resources once the rendering finishes (Fig. 5). A promise is a proxy for a future value and commonly used to enable

Easy but static ⟶ **Complex but dynamic**

Pile of images
`[{src: 'img-01.jpg'}]`

Pile of aggregated SVGs
`[{src:'<svg>…</svg>'}]`

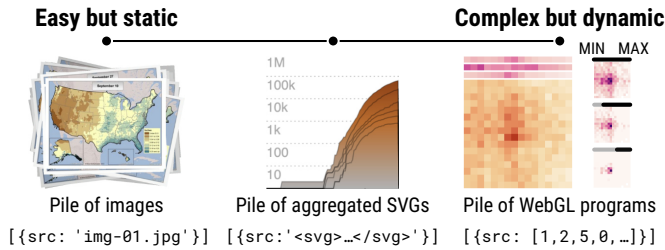Pile of WebGL programs
`[{src: [1,2,5,0,…]}]`

Fig. 6. **Rendering Regimes.** Static renderers are easy to set up but offer limited support for aggregation. Dynamic renderers can be complex but support dynamic updates (e.g., color scaling) and aggregation.

Pile Scale
```
pile ⇒ 1 + Math.min(
    2, (pile.items.length - 1) * .05)
```

Item Brightness
```
(item, i, pile) ⇒ Math.min(
    .85, .05 * (pile.items.length - i-1))
```

Fig. 8. **Dynamic View Properties.** (Left) We dynamically scale the pixel size of the pile by the number of items. (Right) We dynamically adjust the brightness of items, based on the item order. Items with a small index appear brighter. The brightness ranges from -1 (black) to 1 (white).

asynchronous executions in JavaScript. The texture resources must be one of the following media types: an image, canvas, or video element. Since many web-based visualizations render SVGs, PILING.JS provides a predefined SVG renderer that accepts an SVG string or element as input. Using the SVG renderer, it is easy to render any static D3 visualization in PILING.JS. PILING.JS also includes a predefined matrix renderer and supports PixiJS [20] WebGL programs as a renderer for dynamic re-rendering (Fig. 6 right). To support *gallery* previews, PILING.JS implements *meta* renderers, which compose multiple images into a single image. These renderers rely on a representative aggregation approach. For convenience, PILING.JS provides a built-in renderer that composes multiple items into a gallery (Fig. 7). In general, each built-in renderer can be replaced or configured for customization.

Aggregators. To support foreshortened item previewing or pile aggregations, the designer needs to specify an aggregator function for items or piles. The aggregator either receives as input a single item or multiple items and returns a single data source that is subsequently passed to the related renderer. By decoupling renderers and aggregators, both can be reused without having to adjust them. Also, not all types of item previews require aggregation. For synthesizing aggregation, PILING.JS provides a set of predefined matrix aggregators that supports common summary statistics (mean, variance, and standard deviation). For representative aggregation, PILING.JS implements a generic cluster-based approach that employs k-means clustering as the computation is fast enough to not cause noticeable delays. We pick the k items that are closest to the k centroids from k-means as the representative images.

5.2 Pile Encoding via View Properties

PILING.JS offers many view properties to specify the arrangement, previewing, and visual encoding of piles (Supplementary Figure S2). For a complete overview of all view properties, please refer to `https://piling.js.org`. View properties are set via PILING.JS's `set` method, which receives as input the property name and value. In general, there are three types of view properties in PILING.JS, which related to our data model (Sect. 5.1): *global*, *pile*-specific, and *item*-specific view properties. Global properties relate to the entire piling interface and do not change during interactive grouping. For instance, `set('columns',10)` sets the number of columns to 10. In contrast, pile- and item-specific properties can depend on the grouping on the `pile`, i.e., the state of grouping, and therefore support dynamic specifier functions (Fig. 8). For instance, the pile border size could be a function of the number of elements on a pile. Inspired by D3 [9], PILING.JS implements a declarative data-driven interface to dynamic properties by passing the specific `items` and `piles` to the specifier function. Using this approach, the designer only needs to specify how to translate the data object into a property value, while PILING.JS visually renders the
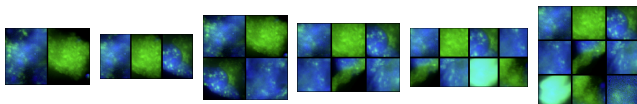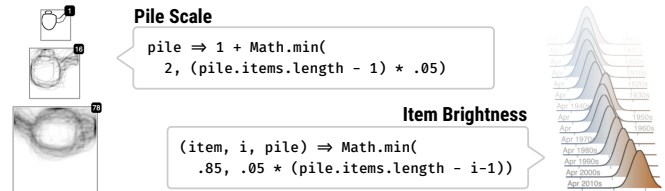
property. Dynamic pile-specific properties are invoked for every pile. They receive the current pile object and return a corresponding property value. Item-specific property specifiers are invoked for every item on a pile as the pile's composition changes. The specifier function receives an item's data object, index, and corresponding pile object, and returns the property value (Fig. 8 bottom-right).

5.3 Pile Interactions

While visual piling is agnostic to the input device (e.g., mouse, pen, or touch), PILING.JS currently focuses on mouse interactions and implements general mouse gestures found across several related works (Sect. 4) for manual grouping, arrangement, browsing, as well as methods for automatic grouping and arrangement.

Gestures for Manual Interactions. PILING.JS implements common gestures for grouping and arrangement (Sect. 4). Piles can be arranged manually via a drag-and-drop gesture. For sequential grouping, an item or pile needs to be dropped onto another item or pile (Fig. 9 top-left). To group multiple items at once, PILING.JS offers a lasso tool. The lasso is initiated by clicking into an empty region of the canvas. Subsequently, a circle will appear (Fig. 9 bottom-left). By clicking into this circle and holding down the primary mouse button, the user can start the lasso selection. All items located within the lasso area are grouped upon releasing the primary mouse button. For browsing (Sect. 4.4), PILING.JS implements gestures for in-place, dispersive, layered, and hierarchical browsing. In-place browsing is triggered by a click on a pile and moving the mouse cursor over the previews. Double-clicking on a pile will temporarily disperse a pile into a regular grid, as shown in Fig. 9 (top-right). To browse a pile in layers, the user can activate the pile's context menu (Fig. 9 top-right) and select "browse separately." The browsed pile is additionally dispersed on the next layer to support rapid sub-piling.

Automatic Grouping and Arrangements To support automatic grouping and arrangement, PILING.JS features a `groupBy`, `splitBy` and `arrangeBy` method. The `groupBy` method enables layout-, proximity-, and data-driven groupings, as described in Sect. 4.1. As a complement, the `splitBy` method can split piles based on their position and data properties of items. Finally, the `arrangeBy` method enables automatic pile-specific arrangements. All three methods rely on a `type` and an `objective`. The `type` determines the subroutine to be used and the `objective` provides the necessary data to execute this subroutine.
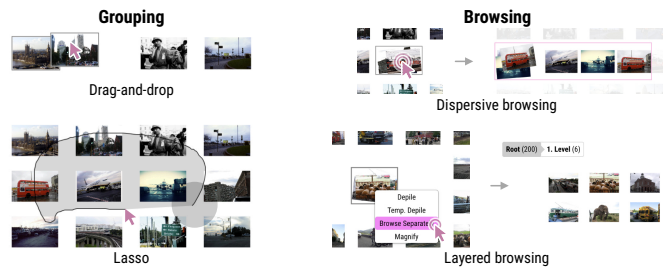

Fig. 9. **Interaction Techniques.** PILING.JS supports drag-and-drop and lasso gestures for manual grouping. In PILING.JS, piles can be dispersed temporarily via a double click and browsed in layers via the context menu.

Fig. 7. **Gallery Previews.** The built-in representative renderer supports galleries of two, three, four, six, eight, and nine previews.
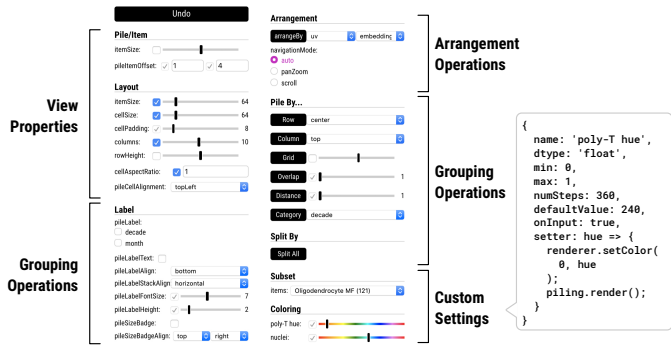
Fig. 10. **Graphical User Interface for Parameterization.** PILING.JS implements several default settings and allows to define use-case specific settings. See Supplementary Figure S4 for a scaled-up version.

For an example, see Supplementary Figure S3. Currently, `groupBy` supports proximity-based (`distance` and `overlap`), layout-driven (`grid`, `column`, and `row`), and similarity-based (`category` and `cluster`) grouping. For instance, `groupBy('category', 'country')` will group all items of the same country, assuming that the `items` contain a property called `country`. We chose this API style to keep the number of public API methods small. The `splitBy` method supports the same proximity- and similarity-based subroutines for splitting piles. The `arrangeBy` offers coordinate-based (`xy`, `ij`, `uv`, and `index`) and data-driven (`data`) layouts, e.g., `arrangeBy('data', 'size')` will order items by a property called `country`. Finally, the proximity-based `groupBy` subroutines can be re-evaluated automatically upon zooming, as the proximity between items might have changed (Fig. 17). Similarly, the `arrangeBy` subroutines can be re-evaluated automatically after grouping piles.

5.4 Adjust and Explore the Piling Interface via a GUI

PILING.JS provides a mid-level API, which hides the state and rendering aspects of visual piling but relies on the designer to implement the rendering pipeline programmatically. As we worked on the use cases (Sect. 6), we realized that switching between a text editor and the browser to parameterize view specification can be time-consuming since the visual feedback is delayed until the browser refreshes. Therefore, as shown in Fig. 10, we provide a simple yet effective GUI to allow the designer to adjust various view properties dynamically.

Currently, the GUI features elements for adjusting static property values such as Boolean flags, single or multiple selections, or numerical values. The GUI also has support for triggering groupings and arrangements. Finally, given the breadth of view properties, it is infeasible to cover every possible setting. Therefore, PILING.JS allows the designer to specify custom settings (Fig. 10 bottom-right).

5.5 Implementation

PILING.JS is implemented in JavaScript using PixiJS [20] for WebGL rendering. We chose PixiJS for its highly-optimized 2D texture rendering and flexible mid-level API, which greatly simplifies the development of WebGL programs. The source code of PILING.JS is free and openly available at `https://github.com/flekschas/piling.js` and features extensive documentation (`https://piling.js.org/docs`) for all available view configurations.

5.6 Performance Evaluation

PILING.JS is designed to support datasets of up to a few thousand items. In the following, we evaluate the initialization time and frame rate (Fig. 11). The initialization time includes the library's startup time, data rendering, and item creation. We compare the time for loading items of the following media types: images, canvas-derived textures, and WebGL programs. For the frame rate, we compare navigation, arrangement, and grouping animations (i.e., animated transitions of the piling state triggered by scripted interactions) using the example

from Fig. 1. Specifically, we examine scrolling, pan-and-zoom, automatic arrangements of all items, and lasso-based grouping, which cover the essential core interactions. We repeated each animation ten times and measured the duration in seconds and frames per second (FPS) in Chromium (v80) on a 2016 MacBook Pro.
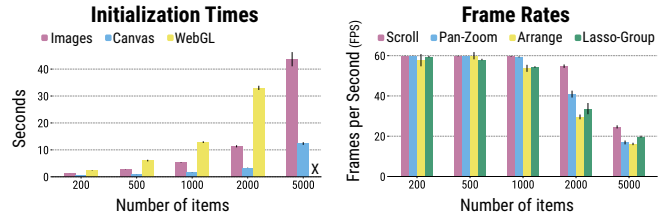

Fig. 11. **Performance Evaluation.** Construction time (smaller is better) and frame rate (higher is better; 60 FPS is the best) as a function of the number of items for different media and interaction types. Loading 5,000 dynamic WebGL renderers ("X") was not feasible as the browser timed out. Error bars show standard deviation.

As shown in Fig. 11, the initialization time increases with the number of items but remains acceptable until 1,000 items. The media type and size of the items greatly influences the initialization time. Especially the custom WebGL programs take longer to initialize. The frame rates for scrolling and pan-and-zoom interactions with datasets of up to 1,000 items is smooth but starts to degrade significantly for datasets larger than 2,000 items. As the arrangement and grouping animations are more involved, their frame rates are lower but remain acceptable for up to 2,000 items.

6 USE CASES

In the following section, we present several use cases to demonstrate the generality of the visual piling approach and the expressiveness of our PILING.JS library for exploring large collections of small multiples. The use cases are also available online at `https://piling.js.org`.

Compiling Training Data for Machine Learning.
A critical aspect of machine learning research, especially deep learning, is the composition of training and validation datasets to probe machine learning models. For instance, in computer vision research, this involves collecting, sorting, and selecting images. While several collections of annotated images exist for comparison and benchmarking [13,16,29,33], subsets are often used during the initial development of the model for exploration. Creating these subsets is not trivial as nuanced image features might not have been extracted, and formal categorization of every potential interesting feature is prohibitive. Visual piling can address this issue by allowing users to sort existing datasets into subsets (**T1**) for rapid hypothesis testing. For example, in Fig. 12.1, we sampled 2,000 images of cars in context from the Microsoft COCO [33] dataset that can be used to train car detector models. Browsing all images provides a first overview. Using the given object
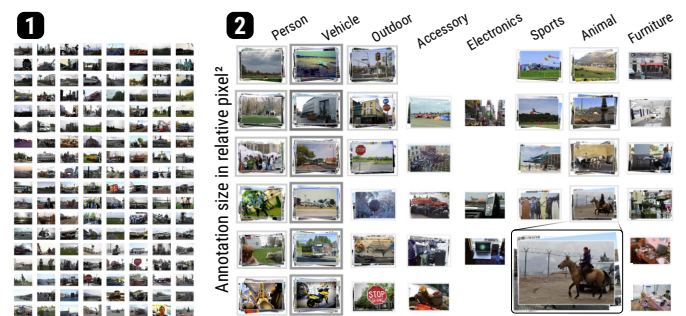

Fig. 12. **Compiling Natural Images from the COCO dataset [33].** All images (1) contain car annotations, but only a subset of them show a car prominently. (2) Arranging the images by their primary annotation type and relative annotation size improves the explorability.
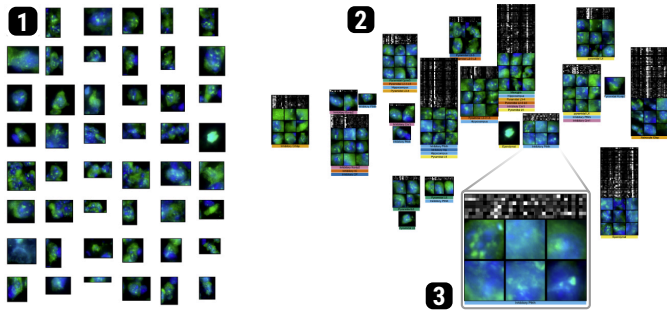
Fig. 13. **Microscopy Cell Instances.** Small multiples of different cell types (1) from an immunofluorescence microscopy image [12] where green shows total mRNA and blue represents nuclei. Visual piling (2) allows for simultaneous exploration of the cell phenotype and related gene expression profiles (black and white heatmap) as highlighted in (3).

Fig. 15. **Annual Precipitation Cycle.** Displaying movie frames as small multiples (1) does not uncover similar scenes well. After arranging the frames by their similarity (2) to show clusters and grouping the frames into piles, we can identify nine distinct states (3).

annotations, we can arrange (**T2**) the images in a two-dimensional grid by their primary category (x-axis) and relative size of the annotation in pixels^2 (y-axis). Google Facets [21] allows for similar arrangements but requires zooming as the number of items increases. With PILING.JS we can instead group all items that are located within the same grid cell into piles, which provides visual cues about the groups' content and the ability to compose new groups manually.

Exploring Instance Annotations in Large Images. One aspect of analyzing large image data involves the exploration of instance annotations. For example, in cell biology, researchers annotate cell boundaries in immunofluorescence microscopy data of tissues or cell cultures. The goal of visual exploration is to compare and organize cells to each other for quality control and stratification (**T1**). Using a conventional small-multiples approach can be limiting when there are several potentially-interesting arrangements. In Fig. 13 we show an exploration of a microscopy image from Codeluppi et al. [12]. Since the cells were clustered based on their gene expression profiles, we arranged cells by the gene expression data reported in the original paper (**T2**). As the cell bodies do not align well, we show a gallery preview of representative images (Fig. 13 2 and 3) as the pile cover to highlight the diversity of cell images across the pile (**T5**). Additionally, we preview individual cell annotations as one-dimensional heatmaps (**T3**) above the cover, which show the cells' gene expression profiles. This enables us to correlate the cell morphology to the gene expression data.

Comparing Repeated One-Dimensional Measurements. Comparing one-dimensional repeated measurements with small multiples typically involves the alignment of items along a shared axis to discover patterns (**T2**). For large numbers of repeated measurements, it can be beneficial to explicitly group measurements to emphasize trends and to interactively change the grouping to highlight different patterns between subsets of the data. For example, in Fig. 14, we loaded the global surface temperature anomaly dataset from NASA [18, 32]. This dataset contains surface temperature measurements for each month across 14 decades (the 1880s to 2010s) that is normalized by the mean temperature of 1951-1980. We plotted the mean temperature deviations

from -1.5 to +1.5 degrees Celsius for each month of the 14 decades (Fig. 14.1). Grouping the plots by decades or months, and arranging them by a vertical offset enables us to dynamically create ridge plot-like piles. Positioning the piles next to each other makes it easy to compare decennial (Fig. 14.2) and monthly trends (Fig. 14.3). We can now immediately see how the temperature increased over the last 140 years.

Movie Analysis. When analyzing movies, it can be insightful to study the visual similarity of scenes. To compare the similarity between frames, Bach et al. [4] folded a linear curve, called a time curve, in 2D space using a dimensionality reduction technique. In Fig. 15.1, we loaded 365 frames from a movie showing the annual precipitation cycle of the United States [11] (one frame per day). Based on the similarity between each frame, we embedded the frames into a two-dimensional space with UMAP [38] (Fig. 15.2). After arranging the frames by their embedding (**T2**), we can highlight the annual precipitation cycle and several clusters of highly similar frames. Visualizing the frames as thumbnails shows what these clusters represent. As the high number of frames makes it hard to compare individual frames, we grouped overlapping frames into piles to simplify the view (**T1**), which highlights nine visually distinct precipitation patterns (Fig. 15.3). Additionally, we encode the frame order via the border color (which ranges from light gray (January) to black (December)) and connect the piles with a line visualization to foster the connection to the underlying sequence of the movie. This line visualization is realized with D3 [9] and linked to the pile interface.

Time Series Analysis. When dealing with time series, an important task is to identify overall trends and variations. To see and make sense of any trends, one must be able to compare individual items. Visual piling can address this challenge through content-aware browsing (**T4**). In Fig. 16, we plot the fertility rate (x-axis) against life expectancy (y-axis) from Worldbank [52] from 1960 to 2017 as small multiples, resolved by country and colored according to the geographic region. After grouping (**T1**) European countries (Fig. 16.2), we can see that, over time, the fertility rate lowers while the life expectancy increases, as shown by color gradient going from bright (1960) to dark (2017). To support comparing individual years without having to split the groups,
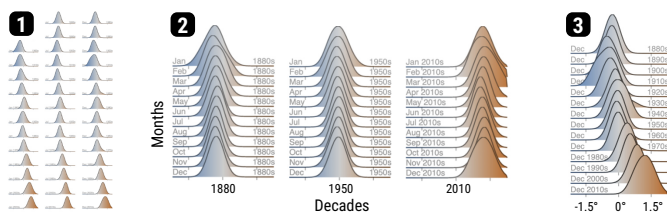

Fig. 14. **Global Temperature Anomalies.** Surface temperature deviations from -1.5 to +1.5 degrees Celsius across decades. Piling enables the dynamic creation of ridge plot-like piles (2 and 3) from small multiples (1) for interactive comparison of decennial (2) and monthly (3) trends.
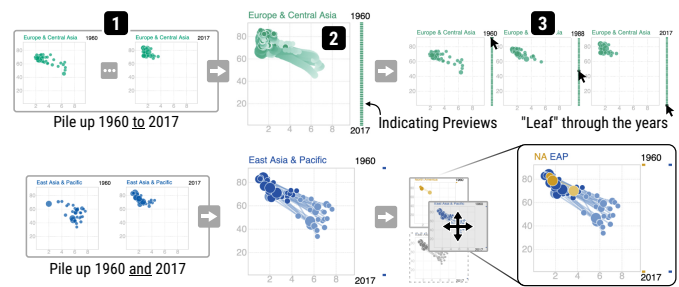

Fig. 16. **Fertility Rate vs. Life Expectancy.** We group small multiples (1) into a pile (2) to compare changes over time, which we browse by leafing through the indicating previews (3). Manual grouping (bottom) allows us to compare East Asia (blue) against North America (yellow).
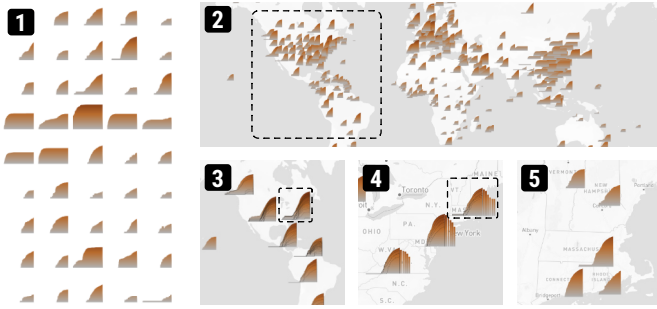
Fig. 17. **Worldmap of COVID-19 Infection Rates.** Small multiples of area charts (1) show the number of infected people over time. Arranging the charts geographically (2) and grouping them by overlap (3) highlights infection hot spots without overplotting issues. Upon zooming in, piles are automatically split (3-5).
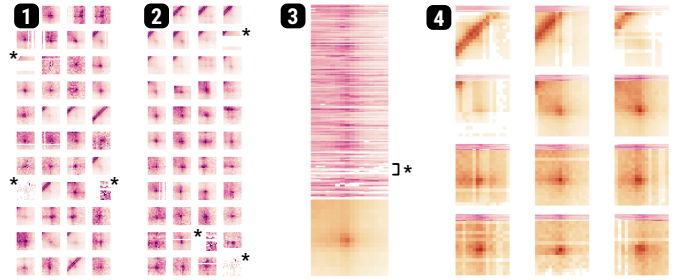

Fig. 18. **Comparison of Matrix Patterns.** Small multiples of matrix patterns (1) that are supposed to show a dark dot in the center. Ordering (2), aggregating (3), and grouping (4) through visual piling enables us to discover overall trends and outliers (*).

we render small rectangles next to a pile as indicating previews (**T3**) of the years. By leafing through the rectangles (**T4**), the year's corresponding scatterplot is revealed, which allows us to trace the temporal development (Fig. 16.3 top). Upon manually grouping scatterplots, we aggregate (**T5**) the data into a combined and connected scatterplot [22], which is shown as the pile cover, to allow tracing the development of individual countries. By piling up the years 1960 and 2017 of North America and East Asia, we can see the alignment of countries in both regions (Fig. 16.3 bottom).

Pattern-Driven Navigation in Multiscale Visualization. A common challenge in exploring local patterns in multiscale visualization is the lack of visual details at an overview [26]. These details are often needed to decide which region to explore in detail. Lens techniques can be applied to magnify a selected region, but many lens techniques do not scale well to large numbers of local patterns. As shown in Scalable Insets [30], the scalability issue can be addressed by displaying local patterns as insets and grouping those insets into piles upon zooming out. In Fig. 17.1, we show area charts of COVID-19 infection rates as small multiples. By arranging the small multiples according to their geolocation (**T2**) we can gain an overview of the global spread (Fig. 17.2). To avoid issues of overplotting, we group (**T1**) overlapping items into piles (Fig. 17.3). Piles are visually represented by a stacked area chart to show the overall regional spread of the virus (**T5**). The combination of grouping and aggregating provides guidance without introducing severe occlusion. Browsing individual countries (**T4**), states, or counties is realized by navigating to a specific area. Upon zooming in, piles are automatically split when the items do not overlap anymore, given their original position (Fig. 17.4).

Matrix Pattern Comparison. A common task in analyzing network data involves the detection and assessment of reoccurring pattern instances, known as motifs. When the data is visualized as a matrix, these motifs can be represented as small multiples. In analyzing motifs, single instances provide only limited insight. Instead, the analyst typically needs to compare individual motifs to groups of motifs. Using a piling interface, this comparison can be achieved by interactively grouping and aggregating the patterns into piles. For example, Fig. 18 shows pattern instances from Rao et al. [39], which should show a dark dot in the center and act as proxies for specific biological events. As these instances are retrieved computationally, the goal is to verify if the expected pattern is truly exhibited. Scanning over the small multiples sequentially is time-consuming but highlights differences (Fig. 18.1). Ordering the small multiples (Fig. 18.2) helps to find instances with the expected pattern (**T2**). By aggregating (**T5**) all instances and showing the average as the pile cover, we can confirm that, on average, the algorithm works as desired (Fig. 18.3). However, by additionally showing one-dimensional previews on top of the cover, we can identify many outliers, which should be removed prior to subsequent analyses (Fig. 18.3 asterisk). Interactive grouping also enables us to stratify (**T1**) the pattern collection for more efficient data cleaning (Fig. 18.4).

7 DISCUSSION & FUTURE WORK

The piling metaphor is a generic and flexible approach for organizing and exploring small multiples. We have demonstrated a variety of use cases, but visual piling can be useful for any small-multiple setup that benefits from interactive re-organization and comparison of the items. This is, for instance, the case when different arrangements can be insightful or multiple aggregated items are to be compared. However, to drive new insights, visual piling requires at least some user interaction.

With the development of PILING.JS our goal is to provide a domain- and application-independent toolkit that is extensible to many different use cases. To this end, we decoupled the rendering and aggregation methods in PILING.JS to allow the designer to provide their own solutions. Depending on the data and visualization type of the items, the visual piling approach can potentially scale to tens of thousands of items given the application of scalable arrangement, browsing, and aggregation methods. For example, to visually scale to piles with many items, the designer needs to choose a previewing layout that either naturally limits the number of previewed items, e.g., partial, gallery, or combining) or applies a preview aggregator that limits the number of previewed items. Technically, to improve the initialization time of PILING.JS for large datasets, we are planning to bundle multiple items into larger textures first and upload the textures onto the GPU in batches. To address the rendering performance, we will optimize the interaction handling. As non-interactive rendering scales to 10,000 and more items, dynamically enabling and disabling interaction handling could greatly improve the performance.

The intended target users of PILING.JS are visualization designers who are familiar with web development. While PILING.JS greatly simplifies the creation of piling interfaces, the user still needs to understand and wrangle the source data to design an effective visualization for the individual small multiples. From our experience in implementing the eight use cases, we learned that designing a piling interface with PILING.JS requires a holistic approach during the development of the different design dimensions, as the dimensions are not independent.

Our structured design space definition provides a framework to guide the design of new interactive visual piling interfaces. In combination with our PILING.JS toolkit, we hope that this work will increase the accessibility of visual piling for a wide range of use cases. Beyond the application of piling to new domains, one exciting direction for future work is the evaluation of the perceptual effectiveness of different previewing approaches. Similarly, many different pen, touch, and mouse gestures can be employed to trigger grouping, arrangement, and browsing interactions. We hope that PILING.JS will streamline user studies development to understand the performance of different piling techniques better.

REFERENCES

[1] A. Agarawala and R. Balakrishnan. Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI âĂŹ06, pp. 1283–1292. Association for Computing Machinery, 2006.

[2] D. Aliakseyeu, A. Lucero, and S. Subramanian. Interacting with piles of artifacts on digital tables. *Digital Creativity*, 18(3):161–174, 2007.

[3] B. Bach et al. Small MultiPiles: Piling time to explore temporal patterns in dynamic networks. *Computer Graphics Forum*, 34(3):31–40, 2015.

[4] B. Bach, C. Shi, N. Heulot, T. Madhyastha, T. Grabowski, and P. Dragicevic. Time curves: Folding time to visualize patterns of temporal evolution in data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):559–568, 2016.

[5] D. Bauer, P. Fastrez, and J. Hollan. Computationally-enriched 'piles' for managing digital photo collections. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*, VLHCC âĂŹ04, pp. 193–195. IEEE, 2004.

[6] D. Bauer, P. Fastrez, and J. Hollan. Spatial tools for managing personal information collections. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, HICSS âĂŹ05, pp. 104b–104b. IEEE, 2005.

[7] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: Enabling interactive multiple-view visualizations. In *IEEE Visualization*, VIS âĂŹ05, pp. 135–142. IEEE, 2005.

[8] X. Bi, S.-H. Bae, and R. Balakrishnan. Walltop: Managing overflowing windows on a large display. *Human–Computer Interaction*, 29(2):153–203, 2014.

[9] M. Bostock, V. Ogievetsky, and J. Heer. D³ data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[10] I. Boyandin, E. Bertini, and D. Lalanne. A qualitative study on the exploration of temporal changes in flow maps with animation and small-multiples. *Computer Graphics Forum*, 31(3pt2):1005–1014, 2012.

[11] B. Brettschneider. U.S. normal daily precipitation, 2014.

[12] S. Codeluppi, L. E. Borm, A. Zeisel, G. La Manno, J. A. van Lunteren, C. I. Svensson, and S. Linnarsson. Spatial organization of the somatosensory cortex revealed by osmfish. *Nature methods*, 15(11):932–935, 2018.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. IEEE, 2009.

[14] P. DiGioia and P. Dourish. Social navigation as a model for usable security. In *Proceedings of the 2005 Symposium on Usable Privacy and Security*, SOUPS âĂŹ05, pp. 101–108. Association for Computing Machinery, 2005.

[15] G. Ellis and A. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1216–1223, 2007.

[16] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[17] A. Fernandez and A. Bergel. A domain-specific language to visualize software evolution. *Information and Software Technology*, 98:118–130, 2018.

[18] N. G. I. for Space Studies. GISTEMP Team, 2020: GISS surface temperature analysis (GISTEMP), version 4, 2020.

[19] A. Girouard, A. Tarun, and R. Vertegaal. Displaystacks: interaction techniques for stacks of flexible thin-film displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI âĂŹ12, pp. 2431–2440. Association for Computing Machinery, 2012.

[20] Goodboy Digital Ltd. The HTML5 creation engine: Create beautiful digital content with the fastest, most flexible 2D WebGL renderer, 2020.

[21] Google PAIR. Facets, 2020.

[22] S. Haroz, R. Kosara, and S. L. Franconeri. The connected scatterplot for presenting paired time series. *IEEE Transactions on Visualization and Computer Graphics*, 22(9):2174–2186, 2015.

[23] M. R. Jakobsen and K. Hornbæk. Piles, tabs and overlaps in navigation among documents. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, NordiCHI âĂŹ10, pp. 246–255. Association for Computing Machinery, 2010.

[24] T. Jankun-Kelly and K.-L. Ma. Visualization exploration and encapsulation via a spreadsheet-like interface. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):275–287, 2001.

[25] J. Jongejan, H. Rowley, T. Kawashima, J. Kim, N. Fox-Gieg, G. C. Lab, and D. A. Team. Quick, Draw: Can a neural network learn to recognize doodling?, 2020.

[26] S. Jul and G. W. Furnas. Critical zones in desert fog: aids to multiscale navigation. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, UIST âĂŹ98, pp. 97–106. Association for Computing Machinery, 1998.

[27] M. Khalilbeigi, J. Steimle, and M. Mühlhäuser. Interaction techniques for hybrid piles of documents on interactive tabletops. In *CHI âĂŹ10 Extended Abstracts on Human Factors in Computing Systems*, CHI EA âĂŹ10, pp. 3943–3948. Association for Computing Machinery, 2010.

[28] K. Kim, S. Kim, and H.-G. Cho. A compact photo browser for smartphone imaging system with content-sensitive overlapping layout. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, ICUIMC âĂŹ12, pp. 1–8. Association for Computing Machinery, 2012.

[29] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[30] F. Lekschas, M. Behrisch, B. Bach, P. Kerpedjiev, N. Gehlenborg, and H. Pfister. Pattern-driven navigation in 2D multiscale visualizations with Scalable Insets. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):611–621, 2019.

[31] F. Lekschas et al. HiPiler: Visual exploration of large genome interaction matrices with interactive small multiples. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):522–531, 2018.

[32] N. J. Lenssen, G. A. Schmidt, J. E. Hansen, M. J. Menne, A. Persin, R. Ruedy, and D. Zyss. Improvements in the gistemp uncertainty model. *Journal of Geophysical Research: Atmospheres*, 124(12):6307–6326, 2019.

[33] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, ECCV âĂŹ14, pp. 740–755. Springer, 2014.

[34] T. W. Malone. How do people organize their desks? implications for the design of office information systems. *ACM Trans. Inf. Syst.*, 1(1):99–112, 1983.

[35] R. Mander, G. Salomon, and Y. Y. Wong. A âĂIJpileâĂİ metaphor for supporting casual organization of information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI âĂŹ92, pp. 627–634. Association for Computing Machinery, 1992.

[36] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, et al. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH âĂŹ97, pp. 389–400. ACM Press/Addison-Wesley Publishing Co., 1997.

[37] M. Martino, H. Strobelt, O. Cornec, and E. Phibbs. Forma Fluens: abstraction, simultaneity and symbolization in drawings., 2020.

[38] L. McInnes and J. Healy. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv e-prints*, 2018.

[39] S. S. Rao, M. H. Huntley, N. C. Durand, E. K. Stamenova, I. D. Bochkov, J. T. Robinson, A. L. Sanborn, I. Machol, A. D. Omer, E. S. Lander, et al. A 3d map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*, 159(7):1665–1680, 2014.

[40] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko. Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1325–1332, 2008.

[41] T. Ropinski and B. Preim. Taxonomy and usage guidelines for glyph-based medical visualization. In *SimVis*, vol. 522, pp. 121–138, 2008.

[42] D. E. Rose, R. Mander, T. Oren, D. B. Poncéleon, G. Salomon, and Y. Y. Wong. Content awareness in a file system interface: implementing the âĂIJpileâĂİ metaphor for organizing information. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR âĂŹ93, pp. 260–269. Association for Computing Machinery, 1993.

[43] R. Sadana, T. Major, A. Dove, and J. Stasko. Onset: A visualization technique for large-scale binary set data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1993–2002, 2014.

[44] D. Smilkov, N. Thorat, C. Nicholson, E. Reif, F. B. Viégas, and M. Wattenberg. Embedding projector: Interactive visualization and interpretation of embeddings. *arXiv preprint arXiv:1611.05469*, 2016.

[45] C. Tominski, C. Forsell, and J. Johansson. Interaction support for visual

comparison inspired by natural behavior. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2719–2728, 2012.

[46] E. R. Tufte. *Envisioning information*, vol. 126. Graphics Press Cheshire, CT, 1990.

[47] S. van den Elzen and J. J. van Wijk. Small multiples, large singles: A new approach for visual data exploration. In *Computer Graphics Forum*, vol. 32, pp. 191–200. Wiley Online Library, 2013.

[48] A. Vogogias, J. Kennedy, D. Archambault, B. Bach, V. A. Smith, and H. Currant. BayesPiles: Visualisation support for bayesian network structure learning. *ACM Trans. Intell. Syst. Technol.*, 10(1):1–23, 2018.

[49] M. O. Ward. A taxonomy of glyph placement strategies for multidimensional data visualization. *Information Visualization*, 1(3-4):194–210, 2002.

[50] N. Watanabe, M. Washida, and T. Igarashi. Bubble clusters: an interface for manipulating spatial aggregation of graphical objects. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST âĂŹ07, pp. 173–182. Association for Computing Machinery, 2007.

[51] S. Whittaker and J. Hirschberg. The character, value, and management of personal paper archives. *ACM Trans. Comput.-Hum. Interact.*, 8(2):150–170, 2001.

[52] Worldbank. Data: Fertility rate and life expectancy, 2020.