ATOM: A Grammar for Unit Visualizations

Deokgun Park, Steven M. Drucker, Roland Fernandez, and Niklas Elmqvist, *Senior Member, IEEE*

Abstract—Unit visualizations are a family of visualizations where every data item is represented by a unique visual mark—a visual *unit*—during visual encoding. For certain datasets and tasks, unit visualizations can provide more information, better match the user's mental model, and enable novel interactions compared to traditional aggregated visualizations. Current visualization grammars cannot fully describe the unit visualization family. In this paper, we characterize the design space of unit visualizations to derive a grammar that can express them. The resulting grammar is called ATOM, and is based on passing data through a series of layout operations that divide the output of previous operations recursively until the size and position of every data point can be determined. We evaluate the expressive power of the grammar by both using it to describe existing unit visualizations, as well as to suggest new unit visualizations.

Index Terms—Visualization grammar, unit visualizations, declarative specification.

1 INTRODUCTION

Visualization encodes symbolic data into visual structures [1], and arguably the most straightforward way to do this is to use a direct mapping where each data item becomes a unique visual mark. Such visualizations strictly maintain the *identity* of each visual mark and its relation to a corresponding data item. Drucker and Fernandez use the term *unit visualizations* to refer to this family of visualization techniques, and prominent examples of such techniques include unit charts, dotplots, and scatterplots [2]. In contrast, visualizations based on data aggregation—such as barcharts, piecharts, or histograms—merge multiple data items into inseparable graphic entities [3]. While such data abstraction improves the scalability of the visual representation, it surrenders the identity property of the visual marks, making it impossible to distinguish individual data points in the visualization. Maintaining the identity property, on the other hand, allows for many novel interactions not possible using an aggregating visualization, such as querying individual data points, tracking their movement during transitions, and filtering on an item level. While many useful visualizations that exhibit these properties exist, to date, this type of visualization has not yet been classified as a unique category, and their design space has not been systematically explored.

In this paper, we address this gap in the literature by presenting ATOM, a high-level grammar for unit visualizations based on a structured exploration of their design space. ATOM uses a sequence of recursive layout operations that organize the output of previous operations until the size and position of each data point can be determined, as shown in Figure 1. In our implementation, Atom specifications are standard JSON objects that are ingested by the Atom engine and then rendered as Scalable Vector Graphics in a modern web browser.

We validate the ATOM grammar using a two-pronged strategy. First, we use ATOM to replicate existing unit visualizations, such as barcharts, mosaic plots, dotplots, and density plots (Figure 2). This approach demonstrates the *expressive power* of the grammar. Second, we use ATOM to create new unit visualization techniques. This yields a number of previously unknown visualizations that may be useful to explore further, and proves that our grammar also has significant *generative power*.

The remainder of this paper is structured as follows: We first define and discuss unit visualizations and their difference from visualizations that use aggregation. We then review the literature on current unit visualizations and visualization grammars. This leads to our design space of unit visualizations and a grammar for describing them. We validate our work with several examples of existing as well as novel unit visualizations. Finally, we discuss the Atom grammar in contrast to existing visual grammars and derive guidelines for how to best use them. We close the paper with our conclusion and our plans for future work.

- *Deokgun Park and Niklas Elmqvist are with the University of Maryland, College Park, MD, USA. E-mail:{intuinno, elm}@umd.edu.*
- *Steven M. Drucker and Roland Fernandez are with Microsoft Research. E-mail: {sdrucker, rfernandez}@microsoft.com.*

2 AGGREGATED VS. UNIT VISUALIZATIONS

We define *unit visualizations* as visualizations that maintain the *identity* property of its visual marks, i.e., where each visual mark is a unique entity that is associated with a corresponding unique data item. The identity property means that for every data item in the data table, there is a corresponding visual mark in its visualization. While the unit visualization family has not yet been properly categorized in the visualization field, there nonetheless exist several examples of effective unit visualizations, such as unit charts, dotplots, and scatterplots.

Maintaining the identity property can lead to visual clutter for large datasets. To combat this, many visualization techniques are based on data abstraction, such as aggregation, segmentation, or filtering [4]. Instead of maintaining an absolute one-to-one mapping between data items and visual marks, these abstracted or *aggregated visualization* techniques merge multiple data items into visual aggregates that can no longer be separated, and where the identity property thus does not hold. Examples of aggregated visualizations are barcharts, piecharts, and histograms.

In this section, we contrast unit visualizations to aggregated visualizations in an effort to identify the areas in which using a unit visualization can be advantageous. Analogously, we also recognize situations where unit visualizations provide limited utility.

2.1 Strengths of Unit Visualizations

Unit visualizations have the following advantages over traditional aggregated visualizations:

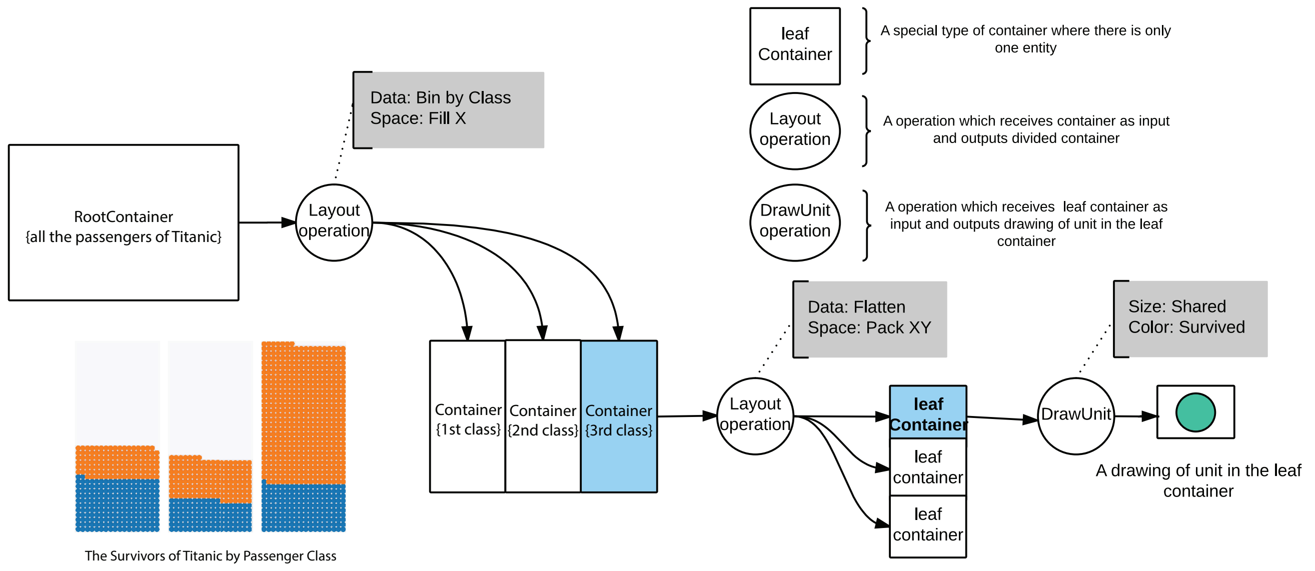
leaf Container
A special type of container where there is only one entity
Layout operation
A operation which receives container as input and outputs divided container
DrawUnit operation
A operation which receives leaf container as input and outputs drawing of unit in the leaf container
Data: Bin by Class
Space: Fill X
RootContainer {all the passengers of Titanic}
Layout operation
Container {1st class}
Container {2nd class}
Container {3rd class}
Data: Flatten
Space: Pack XY
Layout operation
leaf Container
leaf container
leaf container
Size: Shared
Color: Survived
DrawUnit
A drawing of unit in the leaf container
The Survivors of Titanic by Passenger Class

Fig. 1. Sequence of layout operations to generate a unit column chart for survivors of the Titanic by passenger class.

- **Intuition:** The identity property ensures that there is a one-to-one mapping between data points and visual marks, which is a simple bijective function that minimizes the need for the user to consider data abstraction when interpreting the visualization. This also allows detecting outliers in a subgroup. As an example, Figure 2(c) shows Miss Helen Loraine Allison, who was the only child in first and second class to die.
- **Perception:** Maintaining item identity allows for tracking items during animated transitions and interaction. While there is a limit to the number of objects that humans can reliably track [5], [6], this property nevertheless allows a user to follow a selected item during a transition [7] and to get the overall gist of where groups of items are moving. This property can also be used for visual sedimentation [8], where data items are accumulated over time.
- **Constructivism and physicality:** Correlating a unique visual mark with a unique data item conforms to how novices think about and construct visual representations using physical tokens [9].
- **Interaction:** The identity property ensures that users can get details on demand for each individual data item [10]. Furthermore, filtering can be performed on a per-item level, with animations showing visual marks appearing or disappearing from the display.

2.2 Weaknesses of Unit Visualizations

On the other hand, there are disadvantages associated with unit visualizations that should be considered during design:

- **Computational scalability:** A key limitation for unit visualization is the scalability of the hardware platform [3], i.e., the memory, computation, and rendering performance associated with managing unique visual marks for all data items. For truly large datasets, or for hardware platforms with limited capabilities—such as smartphones, tablets, and smartwatches—this can become a limiting factor against adopting a unit visualization.
- **Display scalability:** A unit visualization is only useful if individual visual marks can be distinguished. This means that there is a limit to how small each visual mark can be relative to the screen resolution or physical size of the display it is being visualized on.
- **Perceptual scalability (visual clutter):** Finally, the human visual system is limited in the number of objects that it can perceive [3], let alone track [5], [6]. While clutter reduction is an important research topic in visualization [11], most of these techniques are based on mechanisms that are in direct conflict with the identity property of unit visualizations, including aggregation, sampling, and summarization. While some of these issues concern cases with a large number of data objects, there are also issues with using unit visualizations for a very small number of data objects, where empty space and aliasing can make comparison difficult. For some tasks, such as comparing proportions, aggregated visualizations—such as stacked bar charts—may be superior if there are small numbers of data objects.

3 Related Work

Even though the *term* unit visualizations is somehwat novel, many unit visualizations have been proposed in the past. In this section, we review these techniques and explain why previous visualization grammars are insufficient for describing them. Table 1 gives a representative sampling of visual representations and visualization systems that can be construed as unit visualizations.

3.1 Unit Visualizations

Having a bijective mapping between rows of data and visual marks is arguably the simplest method to generate visualizations in the same sense as how we first learn to represent numbers as children by counting fingers on a hand. A simple extension of finger counting is to use visual shapes to represent data as tallies, where evidence of their use has been found as early as the upper Paleolithic eras. Neurath used multiple repetitive icons to represent quantities of information in his Isotype work in the early 1930's [12]. Waffle charts or square pie charts uses a square matrix and fills the portion

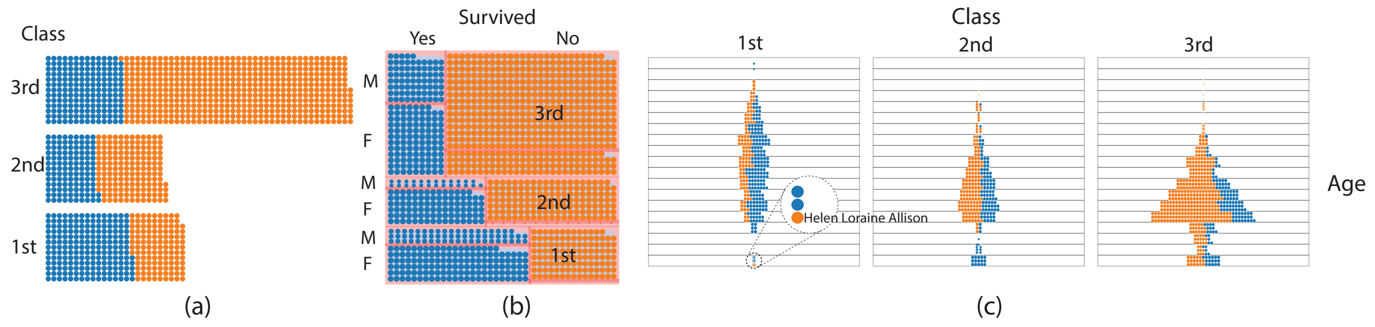

Fig. 2. Example unit visualizations authored using the Atom grammar for the *Titanic* dataset: (a) Unit barchart for passenger class; (b) unit mosaic plot for passenger class, survival, and gender; and (c) unit violin plot for age distribution faceted by passenger class. In each chart, blue dots represent the people who survived and red dots represent those who did not.

grid with colors to show the compositions of data while keeping individual points. More recently, Huron et al. has examined how tokens can be used to help teach basic visualization literacy [9], [13]. However, in these cases, the individual blocks are physical representations of numerical quantities rather than an individual data table row. Previous visualizations have delivered the numerical information effectively for print media. But the development of interactive visualizations leads to a more extreme approach tying visual marks to data. Representing each data row as a visual mark and interactively rearranging them to find patterns have been used in exploratory business analytics in commercial software [2] and in classroom environments [14].

When the dimensions associated with data increases, visualizations can map these additional attributes to visual variables [15] such as position, shape, or area of a visual mark. The simplest version of this, the scatterplot, has been in use since the mid-17th Century. Other examples include bubble charts [16], popularized by Hans Rosling in his Gapminder work.

When *not* directly mapping the position of marks using data, marks can be "packed" onto the screen. Examples of this range from Wilkinson's dotplots [17] to Keim's pixel charts [18]. Much of this work tends to blur the distinction between aggregate-based visualizations and unit-based visualizations, where the units are laid out in a way that reveals both the individual units themselves as well as the overall statistical structure of the data.

While unit representations in static visualizations might be useful for their simplicity, interaction and animation provide opportunities to highlight some of the utility that the individual representation of each row affords [2], [19], [20], [21]. These interactive multidimensional visualization systems maintain object identity during interactive visual exploration such as axis changes, filtering, brushing, and selection. Representative examples of unit visualizations are shown in Figure 3.

Unit visualizations have arisen naturally in the visualization community; therefore, many variations have been developed in an ad-hoc fashion based on heuristics or the authors' intuition. This paper extracts the common factors in these unit visualization representations and creates a general framework so that both the description of existing visualizations and the design of novel ones can take a more systematic approach. As grammars for visualizations have been successful in unifying many disparate visualization types [22], we use a similar approach for unit visualizations with the goal of achieving more formal mathematical rigor.

3.2 Grammars for Visualizations

Visualizations can be constructed using tools at various levels of abstraction. To support programming visualizations from scratch, many libraries have been proposed that provide basic primitives, including Prefuse [31], Processing [32], D3 [33], and Protovis [34]. However, programming falls outside the reach of many people, and requires undue focus on implementation details rather than freeing the designer to focus on the visual representation. Furthermore, new visualizations cannot as a rule be enumerated using a general-purpose programming language.

Declarative languages decouple the specification from the execution [35], [36], and using a declarative visualization grammar allows for simpler description as well as enabling enumeration of legal visual representations. Many declarative visualization grammars have been introduced with distinct goals: some have lower levels of abstraction allowing more expressiveness, while others offer more simplicity. Examples of these pure declarative methods include ggplot2 [37], ggvis [38], Vega [36], Reactive-Vega [39], and Vega-Lite [40].

One of the first examples of this grammar-based approach to visualization was Wilkinson's "Grammar of Graphics" (GoG); an abstraction that makes thinking, reasoning, and communicating about graphics much easier [22]. Building upon these notions, ggplot2 is a widely-used R package for visualizations that implements GoG [37]. However, GoG and ggplot2 are focused on visual specification and do not provide the interaction operations necessary for truly interactive graphics.

Vega [36] extended the specification of visual representations with support for modeling the interaction design. Reactive Vega [39] provided a robust implementation of this in the Vega grammar based on event-driven reactive functional programming. However, even declarative grammars tend to be verbose, making it time-consuming and difficult for novices to construct visualizations. Therefore, Vega-Lite [40] was developed to sacrifice some of the expressiveness of Vega while gaining easier use.

Beyond general-purpose visualization grammars, also domain-specific grammars have been developed for specific types of visualizations. Product plots by Wickham and Hofmann can generate more than 20 statistical graphics by the combination of a few primitives for absolute counts or relative proportions [41]. Baudel and Broeksema [42] describe the design space of sequential space-filling layout, including many variants of treemaps [43], mosaic plots [44], and pixel bar charts [44], with five independent dimensions. MacNeil and Elmqvist [45] propose a view specifi-

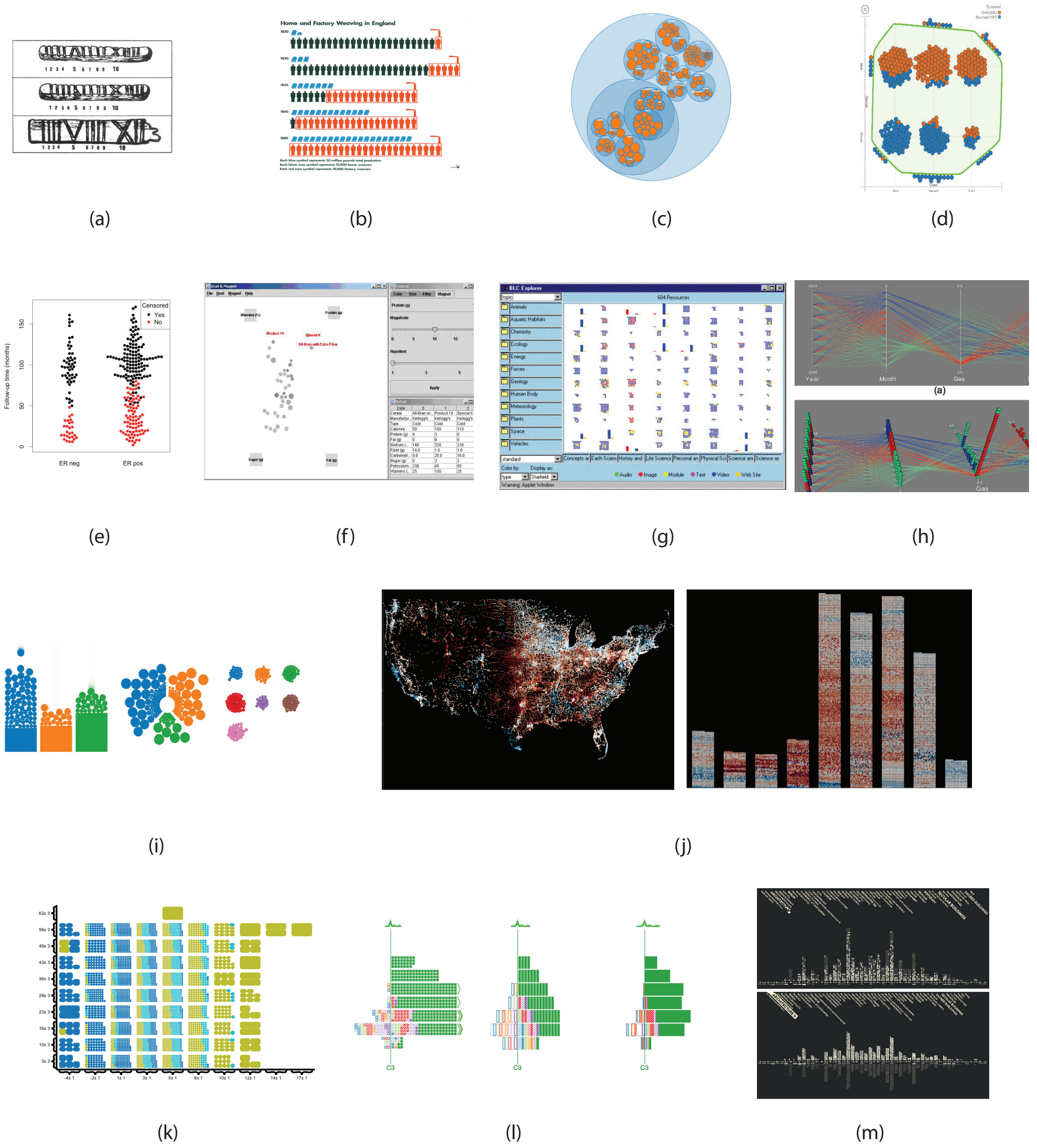

Fig. 3. Examples of unit visualizations. (a) Tallies; (b) ISOTYPES [12]; (c) Circle packing [16]; (d) Kinetica [21]; (e) Beeswarm [23]; (f) Dust & Magnet [24]; (g) Hierarchical axes [25]; (h) Stacker [26]; (i) Visual sedimentation [8]; (j) SandDance [27]; (k) Gatherplots [28]; (l) Squares [29]; (m) Past Visions [30].

cation grammar for slicing and dicing datasets and visual space into mosaics similar to Atom, but their grammar operates at the granularity of individual visualization techniques in the resulting tiles. Overall, there is a tradeoff between the compactness of the domain-specific grammar versus its expressivity. Atom adds several primitives in order to support a wider set of possible visualizations than many of the grammars discussed above.

3.3 Contributions

According to our survey, existing declarative grammars are currently insufficient for describing the class of unit visualizations we have identified in this paper. To date, only low-level programming libraries such as D3 can be used to describe unit visualizations, and this must be done directly at the mark placement level.

The grammar we propose in this work, Atom, incorporates many of the concepts of Wilkinson's Grammar of Graphics [22], such as mark selection, statistical calculation, and aesthetics. However, in the Grammar of Graphics, only collision modifiers such as dodging and stacking are used to avoid the overlapping, which limits the range of potential layouts, whereas Atom provides more sophisticated such operators. Also, in contrast to GoG-style grammars, which tend to specify layouts implicitly based on mark choice, visualizations in Atom emerge from a combination of low-level grouping and primitive layout operations. In this way, Atom has more in common with rule-based layout systems such as graftals and L-systems [46].

4 Design Space of Unit Visualizations

A visualization grammar should be able to express the design space of the visualization type with minimal specification. In this section, we explore the design from the perspectives of visual space, layout, and mark representation. We focus on static representations in this paper, leaving a survey of animation and interaction patterns for unit visualizations as future work.

4.1 Visual Space

The visual space dimension determines the visual coordinate system, and typically includes 1D, 2D, and 3D coordinate systems. It is important to distinguish between data dimensions and visual dimensions. For example, a dotplot represents one data dimension but requires 2D visual space. Similarly, a treemap of a single level uses a single data dimension, but requires 2D visual space. When the input data is 2D, 3D visual space can be used; for example, in 3D dotplots [26].

4.2 Layout

The layout strategy decides the positioning of visual marks for each data object. We identify two fundamental types of layout strategies for unit visualizations: overlapping and non-overlapping layouts.

Overlapping layouts allow visual marks to intersect on the display. This means that the position and shape of the marks are independent, i.e., they can be arranged on the display without considering other marks. A scatterplot, which is one of the most widely used types of statistical graphics, is an example of an overlapping layout [50]. While simple and effective, scatterplots suffer from *overplotting*—where two or more marks partially or completely overlap—when one of the variables mapped to position is categorical or countable. Even when both variables in a scatterplot are continuous numbers, large or locally dense datasets can yield overplotting. Previously, *jittering*—where occluded points are translated from their original positions to become visible—was used for categorical variables [22], [51] to intelligently reduce point occlusion [52], or separated using user-controlled attractors [24]. Overplotting defeats much of the potential benefits of unit visualizations, in that individual points obscure other points, thus preventing judgment about overall numbers and individual interactions. Conceptually, however, overplotting "fits" within a unit specification. In vector file formats such as SVG, the dots are still in the file's contents, even though they are not visible. These hidden units can be revealed by changing the transparency property of each individual marks in some cases when the overlapping is not severe. Even though we allow overlapping layouts such as Mapping (as shown in Figure 4) in unit visualizations, we may wish to avoid overlap and non-overlapping or space-filling layouts are very common in existing unit visualizations.

Non-overlapping (or space-filling [50]) layouts remove overplotting by organizing visual marks disjointly in visual space. However, avoid overlapping requires developing methods to properly partition and organize the output visual spaces. Usually the position and shape of marks depend on the input domain and output range at the same time. For example, given a rectangular unit representation in 2D visual space, the layout algorithm must calculate four parameters for each visual mark—position (x, y) and dimension $(width, height)$—under the constraints that each mark should be disjoint from other marks.

Non-overlapping layouts can be further specialized to reduce the number of possible layouts and enable efficient comparison among visual marks into two common patterns: *subdividing* and *packing*. Subdividing fixes one visual dimension such as width or height to be that of the entire visual space, so that the remaining dimension can be determined by the data property. Because one dimension is maxed out, positioning becomes a trivial matter of sorting. Packing, on the other hand, lays out a visual mark—either a square or circle—in an 1:1 aspect ratio. Squarified treemaps [43] or circle packing [53] are examples of a packing layout. Beeswarm plots [23] enable more efficient, non-overlapping arrangements of points in a scatterplot. Hieraxes [25] avoids overplotting of visual marks by stacking them to resemble fluctuation diagrams. Finally, new packing layouts are based on physicalization, such as the Kinetica [21] and TouchViz [20] systems, which both use visual marks that resemble physical objects by occupying space and being affected by gravity.

4.3 Mark Representation

The visual representation of individual marks can either be a static predefined geometry or used as a visual variable mapped by data properties. Rectangles are the most common representation for non-overlapping layouts in that they require minimal parameters to fix in the visual space and are easy to partition recursively. Because non-overlapping unit visualizations show all the data without overlap, images can be used as marks. Circles also have been commonly used for both overlapping and non-overlapping layouts, and are particularly common in scatterplots and dotplots.

5 Atom: A Unit Visualization Grammar

Above we explored the design space of existing unit visualizations and abstracted the underlying principles, especially the layout operations that differentiate unit visualizations with aggregated visualizations. The grammar we propose in this paper, Atom, is based on this survey, and enables the specification of various visualizations into succinct orthogonal grammar components. Its name, Atom, is obviously derived from the Greek word *atomos*, meaning indivisible, in the sense that a visual mark in unit visualizations are not separable.

Unit visualizations are intended for multidimensional datasets. We can represent a multidimensional dataset as a set $\mathscr{D}$, where each member o is an data object or a row in a data table with attributes $a_1, a_2, \ldots, a_m$, where m is the number of attributes in the dataset. The visual space is a set $\mathscr{V}$, which is composed of all points p in the theoretical space. It is different from a physical canvas on the display, where the visualization is drawn. For example, we can map a small portion of visual space on the whole output display to zoom in a specific part of visualization. We define a *container* $\mathscr{C}$ as a tuple of $(\mathscr{D}_c \subseteq \mathscr{D}, \mathscr{V}_c \subseteq \mathscr{V})$. The *root container* is a container where the data is the entire dataset, and the canvas is the entire visual space. A *cell*, on the other hand, is a container whose set $\mathscr{D}_c$ contains only one element.

A unit visualization operation is an operation that generates a set of subcontainers $\{(\mathscr{D}_{o_1}, \mathscr{V}_{o_1}), (\mathscr{D}_{o_2}, \mathscr{V}_{o_2}), \ldots, (\mathscr{D}_{o_n}, \mathscr{V}_{o_n})\}$ as output given container $\mathscr{C}_i = (\mathscr{D}_i, \mathscr{V}_i)$ as an input, where the data $\mathscr{D}_{o_i}$ and the spatial domain $\mathscr{V}_{o_i}$ satisfies $\mathscr{D}_{o_i} \subseteq \mathscr{D}_i$ and $\mathscr{V}_{o_i} \subseteq \mathscr{D}_o$. Hence, a unit visualization operation is composed of two suboperators in the data domain and spatial domain, respectively. Given the parent

TABLE 1
Classification of existing unit visualizations in the literature.

			Layout						Coordinate System					Unit			
Name	**Author**	**Year**	**Grid**	**Pack**	**Map2D**	**Physicalization**	**Treemap (2D)**	**Fill (1D)**	**Rectangular**	**Cartographic**	**Polar**	**2D**	**3D**	**Geometric shape**	**Icon**	**Images**	**Fig.**
Tallies	Unknown	–	√						√			√		√			3 (a)
Scatterplots	Unknown	1600s			√				√			√		√			
Choropleths	Dupin	1826			√					√							
Stem and leaf	Bowley	1900s	√					√	√			√			√		
Isotypes [12]	Neurath	1936	√					√	√			√			√		3 (b)
Pixel bar charts [18]	Keim et al.	1999	√					√	√			√		√			
Dotplot [17]	Wilkinson	1999		√	√				√		√			√			
Hierarchical axes [25]	Shneiderman	2000	√						√			√		√			3 (g)
Quant. treemaps [47]	Bederson	2001	√				√		√			√		√		√	
Dust and Magnet [24]	Yi et al.	2005				√			√			√		√			3 (f)
Circle packing [16]	Wang	2006		√					√		√			√			3 (c)
Bubble chart [48]	Rosling	2007			√				√			√		√			
ScatterDice [19]	Elmqvist et al.	2008			√				√			√		√			
PivotViewer	MS Livelabs	2009	√					√	√			√		√		√	
Stacker [26]	Dang et al.	2010		√	√					√			√	√			3 (h)
SandDance [2]	Drucker et al.	2012	√		√			√	√	√		√	√	√		√	3 (j)
Histomages [49]	Chevalier	2012	√		√			√	√					√			
Kinetica [21]	Rzeszotarski	2013			√				√			√		√			3 (d)
Constructive vis. [13]	Huron et al.	2014		√				√	√	√		√		√			
Gatherplots [28]	Park et al.	2014	√		√			√	√		√	√		√			3 (k)
Visual sediment [8]	Huron et al.	2014				√			√		√	√		√		√	3 (i)
Phys. token vis. [9]	Huron et al.	2014		√				√	√	√		√	√	√			
Beeswarm plots [23]	Eklund	2015		√	√			√	√			√		√			
Past Visions [30]	Dörk	2016	√					√	√			√				√	3 (m)
Squares [29]	Ren et al.	2016	√					√	√			√		√			3 (l)

container, the data domain operation divides a dataset of parent container into a set of datasets for child containers. Based on our design space, these are the most common such operations:

- `BIN:` Partitioning of $\mathscr{D}$ according to the values of attributes such that all subsequent child groups will contain the different values of an attribute;
- `DUPLICATE:` Duplicate $\mathscr{D}$ into subcontainers;
- `FILTER:` Partition according to a given condition such that one group contains data object that meets the condition and the other group contains the remainder; and
- `FLATTEN:` Partition so each subcontainer has a single item.

The spatial domain operation splits the parent space into child spaces and assigns them to the output datasets of the data domain operation to produce the child containers. We list the most common spatial domain operations in Figure 5.

To produce the target visualizations, our grammar builds a root container and recursively applies unit visualization operations until all containers become cells. In other words, rendering becomes a tree traversal, where the root container is the root of the tree and the cell containers are leaves. Once all cells have been generated, the layout is complete and the visualization can be drawn.

These recursive layout operations are inspired by the layout process of product plots [41]. Atom can be thought of as closely related to product plot because space-filling layouts are quite similar to the composition rule of product plots. However, overlapping plots such as scatterplots are not within the scope of product plots, and Atom adds a cell operation at the end of layout operations, making all data points unique visual marks.

The Atom grammar is formally defined as $\mathbf{G} = (\mathbf{V}, \mathbf{\Sigma}, \mathbf{R}, \mathbf{S})$, where $\mathbf{V}$ is a set of variables, $\mathbf{\Sigma}$ is a set of terminals, $\mathbf{R}$ is a set of production rules, and $\mathbf{S}$ is a start symbol.

Here are the production rules $\mathbf{R}$ in BNF notation [54]:

$$\langle Start \rangle ::= \langle Root \rangle \langle Layouts \rangle \langle Marks \rangle \tag{1}$$

$$\langle Root \rangle ::= DATA\ CANVAS \tag{2}$$

$$\langle Layouts \rangle ::= \langle Layout \rangle | \langle Layout \rangle \langle Layouts \rangle \tag{3}$$

$$\langle Layout \rangle ::= \langle DataOp \rangle \langle VisualPolicy \rangle \tag{4}$$

$$\langle DataOp \rangle ::= \langle BinOp \rangle | DUPLICATE | \langle FilterOp \rangle | FLATTEN \tag{5}$$

$$\langle BinOp \rangle ::= BIN | BIN\ BINSIZE \tag{6}$$

$$\langle FilterOp \rangle ::= FILTER\ CONDITION \tag{7}$$

$$\langle VisualPolicy \rangle ::= \langle VisualOp \rangle \langle Size \rangle \langle isShared \rangle \tag{8}$$

$$\langle VisualOp \rangle ::= MAP2D | FILLX | FILLY | MAXFILL | PACK \tag{9}$$

$$\langle Marks \rangle ::= \langle Size \rangle \langle Shape \rangle \langle Alignment \rangle \langle isShared \rangle \tag{10}$$

$$\langle Size \rangle ::= UNIFORM | \langle SizeFunc \rangle \tag{11}$$

$$\langle SizeFunc \rangle ::= COUNT | SUM\ VAR \tag{12}$$

$$\langle Shape \rangle ::= CIRCLE | RECTANGLE \tag{13}$$

$$\langle isShared \rangle ::= TRUE | FALSE \tag{14}$$

A visualization can be defined by specifying the root container $\langle Root \rangle$, the layout operations, and how individual marks will be represented, as shown in Rule (1). The root container requires the data and the associated canvas for visualization (Rule (2)). Rule (3) states that there can be one or more layouts. Each layout

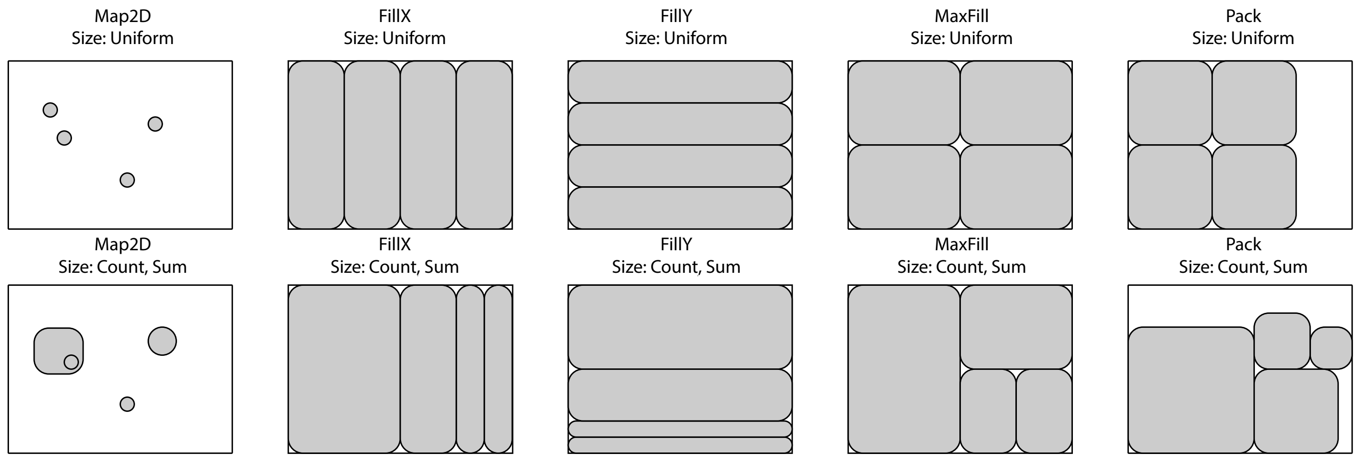

Fig. 4. Visual representations of example operations (VisualOp) and the resulting subcontainers for unit visualizations.

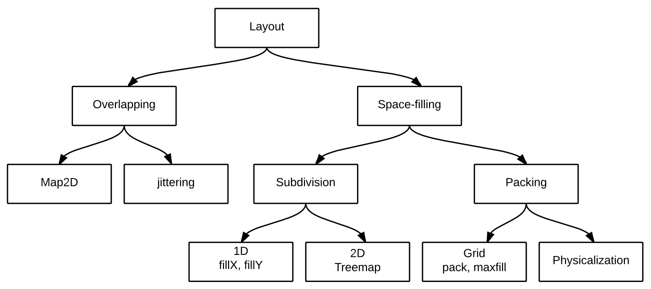

Fig. 5. Common layout operations for unit visualizations.

is composed of a data operation and visual policy (Rule (4)). Available data operations are shown in Rule (5). The *BIN* operation can be performed according to attribute values in the case of nominal variables, and can have an optional *BINSIZE* parameter that indicates the number of bins in the case of continuous variables, as in Rule (6). A visual policy is composed of a visual operation, size setting, and the $\langle isShared \rangle$ setting (Rule (8)). Size can be either uniform or a function of the contents (Rule (11)). Figure 4 shows operations in the Atom grammar with the variable size vs. uniform size. If type is sum, it can have an additional key parameter that will determine the variables to be summed (Rule (12)).

Finally, a visual mark is specified using geometric attributes. The available attributes are simple because a single mark is drawn on a cell container, where there is only one data object.

As a concrete example, we will use the *Titanic* dataset to explain how our unit grammar can draw the unit column chart in Figure 6. The root container is the container, where $\mathscr{D}$ is a set that contains all the people on the *Titanic*, and $\mathscr{V}$ is given as the whole canvas to draw. The `pack` aspect ratio can lead to wasted space, especially when there is a small number of data points inside a container. To resolve this, we use `maxfill` to find an optimal aspect ratio that is close to `pack` while removing empty space.

5.1 Shared Property

Operations in Atom have a sharing setting to control the input domains for the visual variable calculation. Sharing can be applied to both data operations and visual operations. This is needed when a child container does not have all the information to be properly laid out, and information from siblings is required.

As a concrete example, in Figure 7 a container containing 1st Class passengers would like to apply the second layout operation of $(Flatten, PackXY)$. However, it does not know what should be the size of the cell container, because it has only information about the number of 1st Class passengers. To calculate the size of a cell size, it has to know the number of objects in the most crowded container, here 3rd Class passengers. Sharing is a mechanism to synchronize the operation among the sibling containers.

When sharing is enabled, the input domains are shared among siblings. Otherwise, the input domain is limited to the current container. For example, Figure 7 shows that by changing the sharing settings of the size variable, we can create absolute and relative versions of the *Titanic* passengers faceted among class. Sharing settings are hierarchical, meaning that each level can have its own sharing configuration on or off, and these affect to what extent the domain is shared as shown in Figure 8. A useful way to understand the sharing flag is thinking of it as a way to specify whether the container should be "absolute" or "relative" with sibling containers.

5.2 View Composition

Existing grammars, such as Vega-Lite [40] or Grammar of Graphics [22], use a separate view composition algebra to construct multiple views. In Atom, no such view composition operators are needed; we can instead use the existing layout operations to build small multiple charts and multiple views.

More specifically, to generate a faceted chart in Atom, where data is separated across multiple views, we can use the `BIN` operator to partition the data into the separate regions. For a repeated chart, where the same data is replicated across multiple representations, we can use `DUPLICATE` to disseminate the data to all views.

5.3 Implementation Details

Our implementation of Atom is built in JavaScript using D3 [33] and Scalable Vector Graphics (SVG) [55]. The core part of the code is 1,186 lines. A website with more examples and an interactive editor is available at https://intuinno.github.io/unit, and the software is available as open source. Extensibility is an important aspect of

```
{
 "data": "data/titanic.csv",
 "width": 320, "height": 240,  "padding": {..},
 "layouts": [{
  "name": "layout1",
  "type": "gridxy",
  "subgroup": {  "type": "groupby",  "key": "Class" },
  "aspect_ratio": "fillX",
  "size": { "type": "uniform",  "isShared": false },
  "direction": "LRBT",  "align": "LB",
  "margin": { ... },  "padding": {... }
 }, {
  "type": "gridxy",
  "subgroup": { "type": "groupby", "key": "Gender", "isShared": true },
  "aspect_ratio": "fillY",
  "size": { "type": "uniform", "isShared": true },
  }, {
  "subgroup": { "type": "flatten" },
  "aspect_ratio": "square",
  "size": { "type": "uniform", "isShared": true },
  "sort": { "key": "Survived"}
 }],
 "mark": {
  "shape": "circle",
  "color": { "key": "Survived", "type": "categorical" },
  "size": { "type": "max",  "isShared": false },
  "isColorScaleShared": true
 }
}
```

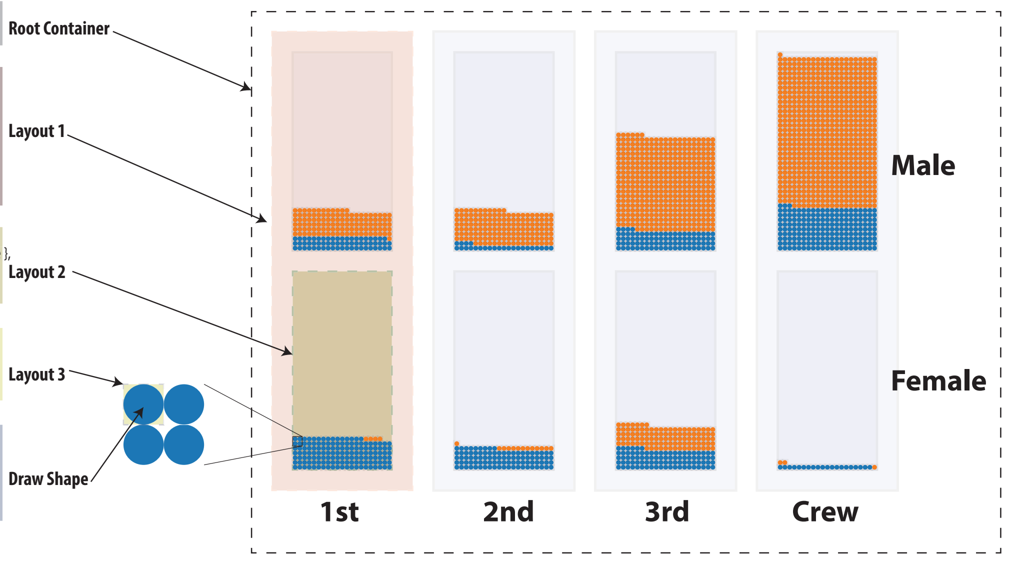

Fig. 6. Example grammar to generate a unit column chart for survivors of the *Titanic* by passenger class.

```
{ ...,
 "layouts": [{
  "subgroup": { "type": "groupby", "key": "pclass" },
  "aspect_ratio": "fillX",
 }, {
  "name": "layout2",
  "type": "gridxy",
  "subgroup": {  "type": "flatten" },
  "aspect_ratio": "maxfill",
  "size": {  "type": "uniform", "isShared": ? }
}]
 "mark": {
}
```

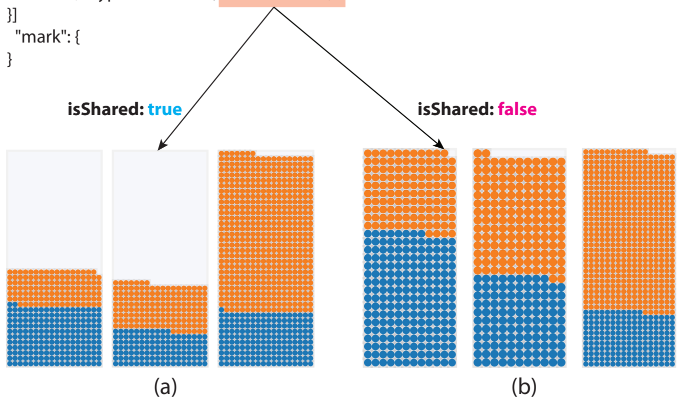

Fig. 7. Unit visualization for the *Titanic* dataset. By varying the sharing flag of the size variable for the second packing layout, we can create both absolute and relative versions. In (a) we can see that the least number of second class passengers survived (absolute count), but (b) shows that third class passengers had the least (relative) chance of survival.

```
{ ...,
 "layouts": [{
  "name": "layout1",
  "subgroup": { "type": "groupby", "key": "gender" },
  "aspect_ratio": "fillY",
 },{
  "name": "layout2",
  "subgroup": { "type": "groupby", "key": "pclass" },
  "size": {  "type": "uniform", "isShared": ? }
  "aspect_ratio": "fillX",
 }, {
  "name": "layout3",
  "subgroup": {  "type": "flatten" },
  "size": {  "type": "uniform", "isShared": ? }
}]
 "mark": {
}
```

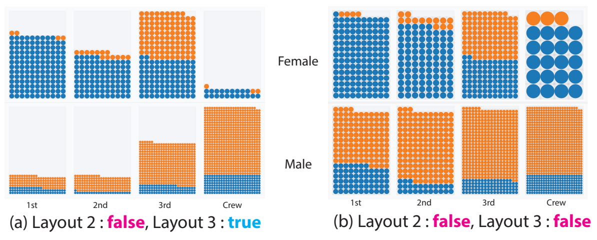

Fig. 8. Sharing can be applied hierarchically. Here the *Titanic* dataset has been faceted by gender and passenger class. In Figure 6, every facet shares the size by setting the size sharing property of "layout2" and "layout3" as true. This yielded a unit bar chart where every dot size is same and the size is adjusted such that the most crowded facet can fill the assigned space. However, (a) shares size only in layout2 that the unit will be the same size among the class but not across genders. This is in contrast to (b), where sizes are independent of gender and class, meaning that every unit will be scaled up to fill their subcontainer.

grammars, where users can add various layout functions such as packing algorithms for polar or three-dimensional coordinates. We leave for future work refactoring of the code base to better support such extensibility.

6 Evaluation

Given our terminology of unit vs. aggregated visualization, we here discuss when unit visualizations are appropriate. We will also discuss their limitations and strategies to overcome these.

6.1 Expressive Power

Table 2 shows how existing unit visualizations can be expressed with grammar components. Basically, we have been able to use Atom to recreate all of the examples in Table 1, with the exception of dotplots [17] since these use a more complex packing algorithm.

6.2 Generative Power

Figure 9 shows a novel visualization that can be generated using Atom. The passengers in the *Titanic* dataset were faceted according

TABLE 2
Expressing existing unit visualizations using Atom.

Type	Operation	Note
Scatterplots	`Map2D`	

Bar+column chart	`FillX` or `FillY`
	`MaxFill`	

Unit pie chart	`FillTheta` `Pack`	Polar coords
Isotypes [12]	`FillX` `Pack`	
Choropleths	`Map2D`	GIS data

Dotplot [17]	`FillX`
	`FillY`
	`Pack`
		Original dotplot not feasible

Hierarchical axes [25]	`FillX`
	`FillY`
	`Pack` (center-align)	

Quantum Treemap [47]	`MaxFill` `Pack`	`MaxFill` with variable size
Bubble Chart [48]	Map2D	Variable size

PivotViewer	`FillX`
	`FillY`
	`Pack`
		Image as mark

SandDance [27]	Multiple operations	
Histoimages [49]	`FillX` (Duplicates) Left (Image): `Map2D` Right (Histo): `Pack`	Colored pixel as mark

Squares [29]	`FillY`
	`Pack` or `FillX`
		Rectangle/square

Past Visions [30]	`FillX` `Square`	Image as mark

to the passenger class and each passenger is represented with a rectangle. The area of the rectangle is proportional to the fare that each passenger paid and sorted so that people who paid the most are located in the top-left. The ratio between the price and the area of the rectangle is shared among child containers so that the comparison between passenger classes is immediately identifiable. The visualization is a combination of unit-based barcharts and a treemap layout succinctly expressed as an ATOM specification.

Similarly, Figure 10 shows another novel visualization where a small variation in the layout specification generates an unit variation of the fluctuation chart.

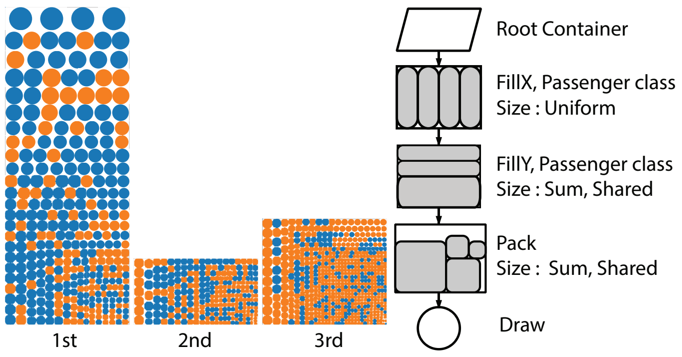

Fig. 9. Unique visualization generated using Atom. Here the passengers of the *Titanic* were faceted according to their class, and then each unit was sized by the price of the ticket.

7 DISCUSSION

Below we discuss some of the finer points of our work on the Atom unit visualization grammar.

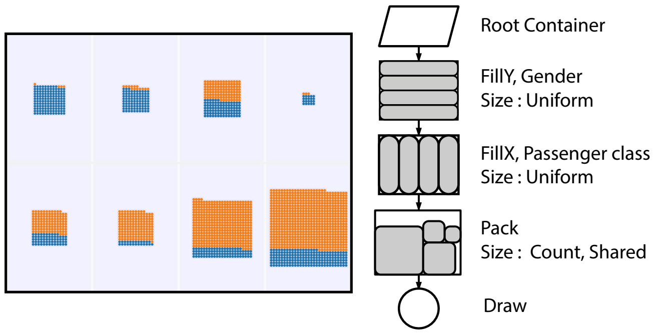

Fig. 10. Unique visualization generated using Atom. Here the passengers of the *Titanic* were faceted according to their gender (vertical) and their class (horizontal), and then each unit was colored based on survival (blue passengers survived).

7.1 Comparison to Existing Declarative Grammars

As mentioned earlier, classic declarative grammars for visualizations, such as those that build upon the work of Wilkinson's Grammar of Graphics [22], are in wide use. In these systems, layouts are typically specified using a combination of mark type (bars, circles, or points, etc.) and a method for mapping attributes of those marks using the data. While we could extend such systems to generate unit visualizations by adding additional layout rules, the way that Atom specifies visualizations is done in a fundamentally different manner: in Atom, visualizations emerge from the recursive application of a small set of primitive rules and layouts at different levels of aggregation. In this manner, Atom shares more in common with parallel rewriting systems such as L-systems and graftals [46], where graphics emerge from the successive application of a series of substitution rules. The difference is that Atom adds the notion of data and aggregate measures to those rules. For example, barcharts or scatterplots are never specified explicitly, but are specified through grouping and packing relationships.

Also, as previously mentioned, product plots [41] are closely related, where a combination of rules produce plots that allow for the visualizations of joint distributions and counts. Since most packing layouts of units are proportional to area, unit visualizations specified in Atom can achieve much of the expressive power of product plots. Similar patterns are found in prior work for domain-specific grammars such as the one proposed by Schultz et al. for treemaps [56]. Their work defines both a theoretical design space as well as a tool for rapid visualization development.

7.2 Exploration of the Design Space

It is intriguing to explore how new types of visualizations might emerge through systematically applying different rules at different levels of aggregation. Figures 9 and 10 show two different visualizations that come from the successive application of those rules. The space of possible combinations, though, is extremely large, and it is beyond the scope of this paper to figure out effective ways for narrowing the enumeration of parameters that produce effective visualizations. Instead we have found that there are certain heuristic situations where unit visualizations are particularly effective in contrast to aggregate visualizations; we list them below. This is by no means an exhaustive list, but our work in this paper lays the foundation for a systematic enumeration of possible visualization parameters.

- **To deliver relative percentage or probability:** When represented in an aggregated visualization, count or relative percentage can be ambiguous in the users' mental model. A classic example is a Bayesian inference problem. For example, a small percentage in a large group can mean much larger absolute numbers than a large percentage in a small group. This concept is notoriously difficult to deliver effectively using text. Garcia-Retamero and Hoffrage found that doctors and patients can make more accurate inferences when information was communicated in natural frequencies rather than probabilities [57]. This is also the underlying problem for the so-called Simpson's paradox [58], where aggregate averages can be deceptive in comparison to the underlying counts.
- **To show underlying distribution of statistical summary:** Wilkinson recommends a tally, stem-and-leaf, or a dotplot as a starting point for analysis instead of commonly recommended bar charts or kernel density estimations [22]. For example, in his book he states that histograms do not reveal granular data, but other unit visualizations do.
- **To show outliers:** Outliers are often lost when using an aggregate visualization because the individual values are averaged in with the rest to produce a single summary statistic. By showing the units themselves, with appropriate attributes such as color and shape applied, outliers and their context can be more easily identified.

7.3 Animation and Interaction

One of the distinct advantages of unit visualizations is that there is a one-to-one correspondence for units in one layout with units in another layout. This allows for straightforward animated transitions when switching between different unit visualizations for the same data by linearly interpolating the positions for each element from its initial and final positions. Staggered starting times [7], [59], staged animations [60], or path clustering [61] can be used to help create more interpretable animated transitions.

Original declarative grammars focused on the generation of static visualizations, as in the case of Grammar of Graphics or ggplot2. However, as interactive visualizations become more common, recent advances allow for describing interactive interactions with declarative specifications. Reactive Vega by Satyanarayan et al. [39] is the first grammar that can specify interactions with declarative specifications. Vega-Lite [40] further simplified the specification by using intelligent defaults and showing novel interactions by enumerating over specifications.

Our Atom grammar currently does not include support for interactivity, but is at this point only a visual specification language, similar to the original Grammar of Graphics [22] or ggplot2 [37]. On the other hand, by focusing on the specific domain of unit visualizations, as Atom does, we can still enable interactions that are common to all unit visualizations, such as item-level selection, details-on-demand, filtering, and cross-highlighting. Further interaction, such as focus+context layouts, advanced navigation, and query operations, are left as future work.

8 Conclusion and Future Work

In this paper, we have defined a new family of visualizations that are based on maintaining the unique identity of each visual mark as well as its direct one-to-one mapping to a data item. Many of these so-called *unit visualizations* are already part of the standard vocabulary of visualization techniques—such as dotplots, mosaic plots, and scatterplots—but our investigation in this paper has revealed that their design space is actually much larger than was previously known. To better capture this new family of techniques, we developed ATOM, a grammar for unit visualizations based on a declarative specification. Our implementation of the Atom grammar can generate any arbitrary unit visualization in this design space. To validate the expressive power of the grammar, we have presented examples of a large number of existing unit visualizations expressed as Atom specifications; to validate its generative power, we have also suggested a number of novel ones.

Our work in this paper is part of a larger trend in the visualization community of abstracting visualizations into high-level declarative grammars. These grammars reduce the need for in-depth programming knowledge and instead enables specifying visualizations in terms of marks, layout, and data. The Atom grammar is specialized for unit visualizations, and it can surely be further refined to support additional visual marks, interaction techniques, and layouts in the future. However, a longer-term research vision should be to find a definitive grammar that can unify many of these existing grammars, while retaining both the simplicity and the power of the original ones.

Acknowledgments

We thank the anonymous reviewers for their helpful reviews. Deok Gun Park and Niklas Elmqvist were partially supported by U.S. National Institutes of Health (NIH) grant R01GM114267. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] S. K. Card, J. D. Mackinlay, and B. Shneiderman, Eds., *Readings in Information Visualization — Using Vision to Think*. Morgan Kaufmann Publishers, 1999.
[2] S. Drucker and R. Fernandez, "A unifying framework for animated and interactive unit visualizations," Tech. Rep., August 2015.
[3] N. Elmqvist and J.-D. Fekete, "Hierarchical aggregation for information visualization: Overview, techniques and design guidelines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 3, pp. 439–454, 2010.
[4] M. C. F. de Oliveira and H. Levkowitz, "From visual data exploration to visual data mining: A survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 3, pp. 378–394, Jul./Sep. 2003.
[5] P. Cavanagh and G. A. Alvarez, "Tracking multiple targets with multifocal attention," *Trends in Cognitive Sciences*, vol. 9, no. 7, pp. 349–354, 2005.
[6] Z. W. Pylyshyn and R. W. Storm, "Tracking multiple independent targets: Evidence for a parallel tracking mechanism," *Spatial Vision*, vol. 3, pp. 179–197, 1988.
[7] P. Dragicevic, A. Bezerianos, W. Javed, N. Elmqvist, and J.-D. Fekete, "Temporal distortion for animated transitions," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2011, pp. 2009–2018.
[8] S. Huron, R. Vuillemot, and J.-D. Fekete, "Visual sedimentation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2446–2455, 2013.
[9] S. Huron, Y. Jansen, and S. Carpendale, "Constructing visual representations: Investigating the use of tangible tokens," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2102–2111, 2014.
[10] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proceedings of the IEEE Symposium on Visual Languages*, 1996, pp. 336–343.
[11] G. P. Ellis and A. J. Dix, "A taxonomy of clutter reduction for information visualisation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1216–1223, 2007.

[12] O. Neurath, *International Picture Language; the First Rules of Isotype: With Isotype Pictures*. K. Paul, Trench, Trubner & Company, 1936.

[13] S. Huron, S. Carpendale, A. Thudt, A. Tang, and M. Mauerer, "Constructive visualization," in *Proceedings of the ACM Conference on Designing Interactive Systems*, 2014, pp. 433–442.

[14] N. Fitzallen and J. Watson, "Developing statistical reasoning facilitated by TinkerPlots," in *Proceedings of the International Conference on Teaching Statistics*, 2010.

[15] J. Bertin, *Semiology of Graphics*. Madison, Wisconsin: University of Wisconsin Press, 1983.

[16] W. Wang, H. Wang, G. Dai, and H. Wang, "Visualization of large hierarchical data by circle packing," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2006, pp. 517–520.

[17] L. Wilkinson, "Dot plots," *The American Statistician*, vol. 53, no. 3, pp. 276–281, 1999.

[18] D. A. Keim, M. C. Hao, U. Dayal, and M. Hsu, "Pixel bar charts: a visualization technique for very large multi-attribute data sets," *Information Visualization*, vol. 1, no. 1, pp. 20–34, 2002.

[19] N. Elmqvist, P. Dragicevic, and J.-D. Fekete, "Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1539–1148, 2008.

[20] J. M. Rzeszotarski and A. Kittur, "TouchViz: (multi)touching multivariate data," in *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems*, 2013, pp. 3119–3122.

[21] ——, "Kinetica: naturalistic multi-touch data visualization," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*. ACM, 2014, pp. 897–906.

[22] L. Wilkinson, *The Grammar of Graphics*. Springer Science & Business Media, 2006.

[23] A. Eklund, *beeswarm: The Bee Swarm Plot, an Alternative to Stripchart*, 2016, R package version 0.2.3. [Online]. Available: https://CRAN.R-project.org/package=beeswarm

[24] J. S. Yi, R. Melton, J. T. Stasko, and J. A. Jacko, "Dust & magnet: multivariate information visualization using a magnet metaphor," *Information Visualization*, vol. 4, no. 3, pp. 239–256, 2005.

[25] B. Shneiderman, D. Feldman, A. Rose, and X. F. Grau, "Visualizing digital library search results with categorical and hierarchical axes," in *Proceedings of the ACM Conference on Digital Libraries*, 2000, pp. 57–66.

[26] D. T. Nhon, L. Wilkinson, and A. Anand, "Stacking graphic elements to avoid over-plotting," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1044–1052, 2010.

[27] "Sanddance," https://www.microsoft.com/en-us/research/project/sanddance/, accessed: 2017-09-30.

[28] D. G. Park, S.-H. Kim, and N. Elmqvist, "Gatherplots: Extended scatterplots for categorical data," University of Maryland, College Park, Tech. Rep. HCIL-2016-10, 2016.

[29] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams, "Squares: Supporting interactive performance analysis for multiclass classifiers," *IEEE Transactions on Visualization and Computer Graphics*, no. 1, pp. 61–70, 2017.

[30] K. Glinka, C. Pietsch, C. Dilba, and M. Dörk, "Linking structure, texture and context in a visualization of historical drawings by Frederick William IV (1795-1861)," *International Journal for Digital Art History*, no. 2, 2016.

[31] J. Heer, S. K. Card, and J. A. Landay, "prefuse: a toolkit for interactive information visualization," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2005, pp. 421–430.

[32] C. Reas and B. Fry, *Processing: a programming handbook for visual designers and artists*. MIT Press, 2007, no. 6812.

[33] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.

[34] M. Bostock and J. Heer, "Protovis: A graphical toolkit for visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1121–1128, 2009.

[35] J. Heer and M. Bostock, "Declarative language design for interactive visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1149–1156, 2010.

[36] A. Satyanarayan, K. Wongsuphasawat, and J. Heer, "Declarative interaction design for data visualization," in *Proceedings of the ACM Symposium on User Interface Software and Technology*, 2014, pp. 669–678.

[37] H. Wickham, *ggplot2: elegant graphics for data analysis*. Springer, 2016.

[38] W. Chang and H. Wickham, "ggvis: Interactive grammar of graphics. R Package Version 0.4.2," *http://CRAN.R-project.org/package=ggvis*, 2015.

[39] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer, "Reactive Vega: A streaming dataflow architecture for declarative interactive visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 659–668, 2016.

[40] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-lite: A grammar of interactive graphics," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2017.

[41] H. Wickham and H. Hofmann, "Product plots," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2223–2230, 2011.

[42] T. Baudel and B. Broeksema, "Capturing the design space of sequential space-filling layouts," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2593–2602, 2012.

[43] B. Shneiderman and M. Wattenberg, "Ordered treemap layouts," in *Proceedings of the IEEE Symposium on Information Visualization*, 2001, pp. 73–78.

[44] M. Friendly, "A brief history of the mosaic display," *Journal of Computational and Graphical Statistics*, vol. 11, no. 1, pp. 89–107, 2002.

[45] S. MacNeil and N. Elmqvist, "Visualization mosaics for multivariate visual exploration," *Computer Graphics Forum*, vol. 32, no. 6, pp. 38–50, 2013.

[46] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[47] B. B. Bederson, "PhotoMesa: a zoomable image browser using quantum treemaps and bubblemaps," in *Proceedings of the ACM Symposium on User Interface Software and Technology*, 2001, pp. 71–80.

[48] V. Battista and E. Cheng, "Motion charts: Telling stories with statistics," in *American Statistical Association Joint Statistical Meetings*, 2011, pp. 4473–4483.

[49] F. Chevalier, P. Dragicevic, and C. Hurter, "Histomages: fully synchronized views for image editing," in *Proceedings of the ACM Symposium on User Interface Software and Technology*, 2012, pp. 281–286.

[50] J.-D. Fekete and C. Plaisant, "Interactive information visualization of a million items," in *Proceedings of the IEEE Symposium on Information Visualization*, 2002, pp. 117–124.

[51] J. Tukey and P. Tukey, "Strips displaying empirical distributions: I. textured dot strips," Bellcore Technical Memorandum, Tech. Rep., 1990.

[52] M. Trutschl, G. Grinstein, and U. Cvek, "Intelligently resolving point occlusion," in *Proceedings of the IEEE Symposium on Information Visualization*, 2003, pp. 131–136.

[53] K. Stephenson, *Introduction to circle packing: The theory of discrete analytic functions*. Cambridge University Press, 2005.

[54] D. E. Knuth, "Backus normal form vs. Backus-Naur Form," *Communications of the ACM*, vol. 7, no. 12, pp. 735–736, 1964.

[55] J. Ferraiolo, F. Jun, and D. Jackson, *Scalable vector graphics (SVG) 1.0 specification*. iuniverse, 2000.

[56] H.-J. Schulz, S. Hadlak, and H. Schumann, "The design space of implicit hierarchy visualization: A survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 4, pp. 393–411, 2011.

[57] R. Garcia-Retamero and U. Hoffrage, "Visual representation of statistical information improves diagnostic inferences in doctors and their patients," *Social Science & Medicine*, vol. 83, pp. 27–33, Apr. 2013.

[58] C. R. Blyth, "On Simpson's paradox and the sure-thing principle," *Journal of the American Statistical Association*, vol. 67, no. 338, pp. 364–366, 1972.

[59] F. Chevalier, P. Dragicevic, and S. Franconeri, "The not-so-staggering effect of staggered animated transitions on visual tracking," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2241–2250, 2014.

[60] J. Heer and G. Robertson, "Animated transitions in statistical data graphics," *IEEE Transactions on Visusalization and Computer Graphics*, vol. 13, no. 6, pp. 1240–1247, 2007.

[61] F. Du, N. Cao, J. Zhao, and Y.-R. Lin, "Trajectory bundling for animated transitions," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2015, pp. 259–268.

Deokgun Park received a bachelor in electrical engineering in 2000 and a masters degree in biomedical engineering in 2002 at Seoul National University, Seoul, South Korea. Currently, he is pursuing a Ph.D. degree in computer science at University of Maryland, College Park in College Park, MD, USA.

Steven M. Drucker received a M.Sc. degree from the MIT AI Lab in 1989 and a Ph.D. in 1994 from the MIT Media Lab, both in Cambridge, MA, USA. He is currently a Principal Researcher in the Visualization and Interaction Group at Microsoft Research in Redmond, WA, USA, and an affiliate professor in the Computer Science and Engineering Department at the University of Washington in Seattle, WA, USA. He is a Senior Member of the ACM.

Roland Fernandez works as a researcher and AI School instructor in the Deep Learning Technology Center at Microsoft Research in Redmond, WA, USA. His interests include reinforcement learning, autonomous multitask learning, symbolic representation, AI education, visualization, and HCI. Before coming to the DLTC, Roland worked in the VIBE group of MSR doing visualization and HCI projects, most notably the SandDance project.

Niklas Elmqvist received the Ph.D. degree in 2006 from Chalmers University of Technology in Göteborg, Sweden. He is an associate professor in the College of Information Studies at University of Maryland, College Park in College Park, MD, USA. He is also a member of the Institute for Advanced Computer Studies (UMIACS), Director of the Master of Science in Human-Computer Interaction (HCIM) program, and Director of the Human-Computer Interaction Laboratory (HCIL) at University of Maryland. He is a senior member of the IEEE and the IEEE Computer Society.