Structuring Visualization Mock-ups at a Graphical Level by Dividing the Display Space

Romain Vuillemot, Jeremy Boy

To cite this version:


HAL Id: hal-01560881
https://inria.hal.science/hal-01560881
Submitted on 12 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
We present a principled design approach for creating visualization mock-ups that are informed by only little, or no data. Our approach is inspired by a dialogue between graphic design theory, and parametric toolkit explorations. We propose that mock-ups can be created at the graphical level, by iteratively subdividing the display space, and progressively informing each division with realistic data, as they become known or available.

The information visualization (InfoVis) pipeline [9, 25] (Fig. 1) indicates that the first step in visualizing data should be to structure the data, before moving on to projecting them in visual form, and rendering them on a display. For the purpose of this article, we refer to this as a data-to-display approach to visualization, upstreaming from the data to the final graphic, or interface. While this is the most commonly accepted approach in InfoVis, designers sometimes have to create visualizations without having access to the data they will eventually be displaying. This can happen for a number of reasons, for example when an interface needs to be open and welcome different types of data, when the data are too sensitive to be shared with the designer, or when the data have simply not yet been collected [42]. In other cases, where the data are available, cleaning, structuring, and processing them can require large amounts of time and effort. This greatly complicates the ideation process for designers who may want to try out different concepts—each of which may require reprocessing the data—while attempting to meet organizational requirements, like iterating with users and other stakeholders, and reporting progress with visual mock-ups.

Adopting the data-to-display approach also tends to imply that the interesting properties and trends in the data are unknown—they should be discovered using visualization. However, when a graphic is created to communicate a message, the designer should have an initial idea of what trends to put forward, in order to develop an intent, or a parti pris [42]. The first step is to think about how to convey the message appropriately, before diving deep into the specific properties of the data. Even for more exploratory interfaces, information architects, user-interface, and user-experience designers can develop an idea of what visual and interactive features they want to include for the user, before actually loading data to the interface.

Inspired by these situations, we propose an alternative approach to visualization, based on the progressive division of space. We refer to this as a display-to-data approach, starting with the configuration of the display space, and moving down to the data (Fig. 1, bottom right). While there are many tools available for creating visualizations, most assume the data-to-display approach. They require an understanding and availability of data, which needs to be cleaned and processed to meet very specific formats; and arriving at a convincing and well-structured final graphic or interface can be very time-consuming—especially if several iterations are required. We argue that our display-to-data approach can alleviate this issue, by helping visualization designers create rapid, low-fidelity mock-ups that can progressively be informed by realistic data, as they become available. We explore this approach from two different perspectives. First, from a graphic design perspective, we analyze how the use of grid systems for structuring the display space, and the progressive segmentation of inlaid modules, can inform the design of visualizations. Second, from a parametric toolkit perspective, we explore how different patterns of space partitioning, in combination with the progressive inclusion of data attributes (whether only anticipated or actually known), can help generate a variety of visualization mock-ups. We detail the creation of several of these mock-ups, highlighting the generative power of our approach. We then discuss how it can facilitate rapid explorations of design alternatives. Ultimately, we hope this work will pave the way for higher-level prototyping tools, useful for InfoVis designers, developers, and teachers.

Fig. 1: Visualizations mock-ups are progressively structured starting from the end of the visualization pipeline; with only little, or no data.
A simple solution to creating visualization mock-ups is sketching. When asked to sketch out a dataset, economists lay out, clarify, store, reflect on, and share their ideas with others [29, 52, 53]. Sketches generally appear to be made quickly; they are not entirely novel representations, they are not common, and contain minimal detail, and are intended to suggest, rather than tell [8]. Low-fidelity, “sketchy” prototypes have indeed been found to be more effective at facilitating ideation, and encouraging feedback in the early stages of design processes, than more polished mock-ups [31, 41, 60].

Roberts et al. [42] advocate for the use of sketching in their Five Design-Sheet methodology for InfoVis, in which they provide grounded support for generating, iterating over, and refining visualization mock-ups, from ideation to realization.

Walny et al. [54] have studied the lifecycle of sketches in software development. They have found that sketches are created, iterated upon, copied, archived, and finally discarded. Much of this lifecycle is influenced by changing communication needs. People draw, redraw, annotate, and transform sketches to share different aspects of a design with others. In our experience, this is typically influenced by the progressive inclusion of realistic data attributes, as they become known or available. This resonates with Walny et al.’s call for visualization tools that can help designers transition through various levels of formality, from sketches to more accomplished “diagram” visualizations.

Walny et al. [55] have also studied how sketching can inform InfoVis design. They have explored the diversity of representations people draw when asked to sketch out a dataset, and have studied the relationship between the types of sketches people produce, and their understanding of the data. They have found that most sketches follow a visual continuum from numeracy to abstraction, and a data continuum from simple statements to conjectures and hypotheses about the data.

Overall, sketching has the advantage of being entirely freeform, and does not depend on specialized equipment: a pencil and paper suffice. It also requires little practice or skill [55] to arrive at satisfying and useful results. However, sketches are not dynamic, i.e. they (generally) cannot be updated, and it is usually difficult to capture the subtleties of certain data attributes (e.g., the specific numbers of entries, dimensions, etc.) as the sketch lifecycle progresses—this is often necessary when moving towards high(er)-fidelity prototypes. More importantly, sketches are only renderings of ideas: in of themselves, they do not provide a principled way of mapping data to visual elements (and vice versa) that can be systematically explored.

2.3 Visualization Design Tools

There are many visualization design tools that aim at facilitating the mapping of data attributes to visual marks and variables. Some offer programmatic APIs, like D3 [6] and Vega [44], while others offer GUIs with more or less advanced mapping capabilities, ranging from simple chart templates in Excel, to more expressive data-binding options in Tableau/Polaris [48], Lyra [43], iVisDesigner [40], or Voyager [61]—the latter even provides mapping recommendations to support exploration. Meanwhile, Sketchstories [32] offers a sketching interface that allows users to invoke chart schema using simple gestures.

Despite their diversity, these tools scarcely allow the design process to start at the graphical level. Whether they first require processing data, or placing graphical primitives on a canvas, they are based on the assumption that visualizations should be ‘built up’ from the data (data-to-display approach), rather than ‘structured down’ from the display space (display-to-data). For example, few of these tools offer quick, high level visualization specifications without the constraint of having an initial dataset. Data-driven grids [28], and to some extent Sketchstories, are an exception (and are probably closest to our approach), as they allow designers to first create freeform graphics, before binding data to them.

3 A Graphic Design Perspective

We first explore the creation of visualization mock-ups at the graphical level from a graphic design perspective. We examine how the use of grids and modules can help lay out the different elements that compose a visualization, with little, or no use of data. We begin by introducing the concepts of modularity and grid systems, and we clarify the basic structural lexicon we use. We then illustrate our approach by creating mock-ups of the “double” and “temporal” treemaps, as well as of the treemap barchart, discussed in our motivating scenario. We detail how we progressively divide the display space to create placeholders for
all the visual marks that build up the different visualizations. We also highlight opportunities for including more realistic data along the way. We start with divisions at the canvas level, where we organize the different elements that complement each chart (e.g., titles, legends, labels, chart-space, etc.), before moving down to the chart level, where we organize the different marks that compose the visualizations.

3.1 Modularity and Grid Systems

Modularity is a principle shared by many design disciplines, as well as architecture [1,12]. It is based on the idea that everything should be built using a strict set of rules, which are defined by a system (e.g., a grid system). In graphic design, grids [2, 22, 37] are essentially vertical and horizontal segmentations of a page; they are the “skeleton” upon which all content is laid out. Modules can be placed at the intersections of these segmentations, which determine placeholders for future content. Vertical segments typically control the inner and outer margins of the page, define the columns, as well as the space separating them; while horizontal segments determine the head and foot margins, the depths of columns, and the location of headlines and visual material [22].

Grids and modular layouts can vary in almost an infinite number of ways, and they are useful throughout the whole design process—from initial low-fidelity mock-ups, to higher-fidelity ones, and even to final designs. They enable designers to rapidly try out different ideas, even before the final content is available. As per our motivating scenario, this is often essential for facilitating necessary discussions among a variety of interlocutors—for example, in the case of magazine design, between the designer, the editor, the production director, the advertisement director, and sometimes the journalist(s).

3.2 Basic Structural Lexicon

\[
\begin{array}{c|c|c}
\text{Vertical segmentation} & + & \text{Horizontal segmentation} \\
\hline 
\text{Grid} & = & \text{Grid}
\end{array}
\]

The basis of our approach is the iterative subdivision of the display space. From a graphic design perspective, this translates into a progressive segmentation of that space, which can be vertical, and/or horizontal, the combination of which results in a grid.

Grids are used to place modules. Modules can range from one to any number of cells, both horizontally, and vertically. Grids can contain several modules, and adjacent modules may, or may not be separated by a single column: the gutter.

In turn, each module has its own inner-space, which can be segmented to create nested segmentations, grids, and modules. Reciprocally, any nested segmentation, grid, or module has a parent space. If a module is not segmented, and is used to place content (e.g., visual marks, text, etc.), it becomes a placeholder.

The resulting combination of grids, modules, and placeholders creates a simple black and white visualization mock-up, which complies with the general look and feel of wireframes in other disciplines. This enables discussions to stay focused on the main design issues, rather than on any superfluous embellishments.

1An interesting collection of grid system examples can be found at: http://www.thinkingwithtype.com/misc/Experimental_Typography.pdf

3.3 Segmenting the Canvas Space

![Fig. 3: Root canvas module: laying out the canvas space for a simple line graph, based on proportions of the canvas width and height. Note that here, the visualization is simply considered an element of content, which is why the root module is represented as a placeholder (left).](image)

We begin by segmenting the canvas space to set up the general layout. This will help place the chart, and the different elements that complement it. Layouts usually start at the edges of the page, and progressively move inwards [34]. The proportions of the page determine the grid, and its ratio is repeated in the inlaid modules. This creates a sense of balance, connectedness, and aesthetic appeal [50]. Although it can be argued that screen-based design, like web design, is progressively moving away from this standard, because webpages have no “edges” (content can flow beyond the page fold, both vertically and horizontally), visualizations are still generally set within a fixed canvas. This unlikely to change, as the “overview first” mantra [46] demands that the whole visualization be visible in a single frame of reference.

Fig. 3 shows the layout of a simple line graph. The vertical and horizontal segmentations of the canvas space are respectively proportional to the canvas width and height—with the slight specificity here that the horizontal segmentation is also co-defined by the line-height of the typesetting. This is why there are more horizontal segments than vertical ones. The resulting grid is used to place the root module in the center, which determines the placeholder for the chart space; and the different elements that complement the chart, i.e. the tile and subtitle. The chart space is a single proportional module, since it is defined by the proportional segmentation of its parent space, i.e. the canvas. It is used to render the linegraph.

![Fig. 4: Nested canvas modules within the root module: setting up nine charts in the same canvas space.](image)

In some cases, the root module may need to contain several charts—typically for small multiples. As for every module, its inner-space can be segmented, using nested segmentations, resulting in a nested grid. Fig. 4 shows a nested, proportional, vertical segmentation of the root module, used to create nine nested modules that each determine a placeholder for a unique chart space. Note that the canvas and root module have been extended horizontally in this figure to welcome these nine modules. Fig. 5 further illustrates how nested segmentations may, or may not be identical to those of the module’s parent space.

Finally, Fig. 6 shows how nested segmentations, grids, and modules can be informed by other values than proportions of their parent space,
like random values, or more realistic data. The central mock-up in Fig. 6 typically illustrates the placeholders for the two chart spaces needed to create a “double” treemap: their different sizes help convey the idea that the overall volume of trade will likely be different.

3.4 Segmenting the Chart Space(s)

When the chart placeholder(s) have been laid out, we can segment the chart space(s) to simulate where the different visual marks in the visualization(s) might fit, and how they might relate to each other. Here, we present a series of examples that lead to a mock-up of a treemap barchart. We begin by prototyping a “double” treemap, following up on the layout in the central mock-up in Fig. 6. We then create a simple barchart, which we use to prototype a “temporal” treemap. Finally, we create a grouped barchart, which we use to combine the “double” and “temporal” treemaps into a treemap barchart.

Fig. 7 shows the construction of a “double” treemap, based on the segmentation of the two chart spaces determined in Fig. 6. Each space is first segmented vertically, proportionally to the chart height (nested, proportional, horizontal segmentations). The subsequent nested modules determine the placeholders for each rectangular mark in the treemaps. Note that we could have equally started with horizontal segmentations, followed by nested vertical segmentations (as done in Fig. 9). In addition, while we only apply these operations once to get a rough mock-up of the treemaps, they can both be repeated as many times as there are expected levels in the hierarchical data. This can easily be iterated on, as realistic data become available.

Fig. 8 shows the construction of three simple barcharts, based on the segmentation of a single chart space, and the progressive mapping of data. The space is first segmented vertically, proportionally to the chart width, to create the bar modules. The space is then segmented horizontally to determine the height of each module. The bottom-left mock-up in Fig. 8 uses a proportional segmentation; the bottom-center mock-up uses a random segmentation; and the bottom-right mock-up uses more realistic data. Note that in this case, the proportional segmentation is sufficient to communicate the type of visualization.

Fig. 9 continues Fig. 8. It shows the construction of a “temporal” treemap. Each bar module is first segmented horizontally, proportionally to its own height. Each resulting nested module is then segmented vertically, proportionally to its width (nested, proportional, vertical segmentation). As mentioned for Fig. 7, these operations can be repeated any number of times.

Fig. 10 is similar to Fig. 8, except it includes an extra nested, proportional, vertical segmentation with no gutters for each bar module, to create a grouped barchart.

Fig. 11 illustrates the combination of the “double” and “temporal” treemaps (Figs. 7 and 9) using the grouped barchart (Fig. 10). It shows two variations of a barchart treemap, one starting the treemaps with a horizontal segmentation (top row), the other with a vertical segmentation (bottom row). This simple comparison immediately reveals that starting with a horizontal segmentation is likely to be more legible, as the alternative makes it harder to distinguish between grouped bars. Seeing this early on can help save time later in the design process.

Finally, Fig. 12 shows an overview the whole mock-up creation process, from the segmentations at the canvas level, down to the those at the chart level.
We now turn to exploring the creation of visualization mock-ups at the graphical level from a parametric toolkit perspective. We begin by discussing the mock-ups with their interlocutors. However, while this perspective is informative, it is not entirely practical. All the examples provided in this section were created using a popular graphics editor (Adobe Illustrator). We believe this sets two important drawbacks. The first is that setting up grids does require time, effort, and a good understanding of the graphics editor; and the automation of this process is complicated. This results in an incompressible initial time cost, and the necessity to collaborate with a skilled graphic designer, which prevents an immediate exploration of different visualization ideas. The second is that, when these ideas begin to crystallize, the mock-ups created with graphics editors do not truly afford the progressive inclusion of data. These software generally do not support the binding of data with visual marks [28,49], which prevents from moving on to higher-fidelity mock-ups.

4 A Parametric Toolkit Perspective

We now turn to exploring the creation of visualization mock-ups at the graphical level from a parametric toolkit perspective. We begin by highlighting the lexical differences between this perspective and the previous one, and we introduce a simple visual grammar we use to describe visualization mock-up variations. We then detail the implementation and API of the D3-based toolkit we developed to explore the generative power of our approach.

4.1 Lexicon Abstraction

```
  +  =
```

Space segmentation  Module placement  Space partition

All examples created from the graphic design perspective systematically started with segmentations of the main space, i.e. the canvas or chart space, to create grids, which determined the size and placement of modules. Here, we use an abstracted operation we call partitioning. A partition automatically lays out a defined number of modules, according to a defined layout pattern. This helps simplify the lengthy process of manually setting up a grid, and laying out modules.

As the resulting modules do not strictly follow an underlying grid, we provide them with individual width, height, and padding parameters (instead of gutters). This complies with standard web design practice, where CSS widths, heights, margins and padding are used to determine the size of, and space between elements.

Partitions follow predefined patterns of space segmentations. Basic patterns are horizontal, vertical, and grid. These follow the simple, orthogonal segmentations we used previously. However, grids can also be triangular, diamond-shaped, or hexagonal [45]. While these layout patterns are not particularly common in InfoVis (save perhaps hexagonal patterns), others like treemap, coordinate system, and pack layouts are [45]. We add these to our lexicon to complement our basic rectangular patterns, and to extend our design space. Note that the partitioning operations for coordinate system and pack patterns are different from others, as they focus more on module placement than on space segmentation (as discussed in [45]).

Partitions can be repeated any number of times, creating nested partitions within each module of the mock-up.

Finally, to add expressiveness, and to cater for line-based charts, we created a visual linking operation that connects each module in a partition with an overlaid line.

4.2 Visual Grammar

In addition to our abstracted lexicon, we propose a visual grammar to describe the visualization mock-up variations we create. A basic vocabulary captures the main operations in our approach, as well as the (expected) data attributes they can be applied to. We believe this vocabulary is necessary, as many mock-ups are composite, and can be difficult to name; we also expect it will help highlight parameter patterns across charts. Our vocabulary consists of:

- six partition patterns\(^2\) (based on Fig 13):
  - vertical \[\]
  - horizontal \[\]

\(^2\)Each partitioning operation has a set of parameters, which we describe in the following subsection.
We implemented our basic vocabulary in a parametric toolkit, using D3 data transformation (e.g., d3.nest), and visual presentation (e.g., d3.scale, d3.layout) functions. To facilitate writing these functions, we developed an application programming interface (API), through which we simply pass a set of parameters in the form of a JavaScript object (see Listing 1). The code is available at https://github.com/romsson/d3-gridding; it furthers the effort to serialize chart parameters [23,44]. This toolkit enabled us to easily explore various computation-based designs, using a parametric approach. We used it to generate all the remaining figures in this article, which are also available at a higher resolution in the code repository (e.g., https://romsson.github.io/d3-gridding/example/capture/display.html).

Our grammar then simply consists in sequentially combining operations with data attributes and operators. For example, consider the first mock-up in Fig. 14 (left). Imagine we are expecting to receive an extremely simple dataset, which we have been asked to visualize. We know that this dataset should have no more than four items (I), and three dimensions (D). We decide to first partition the chart space vertically (||) according to the different dimension. We then nest each item horizontally (□) within the resulting modules. This simple succession of operations can be summarized using our grammar as follows: (||D) \times (□I).

### 4.3 Toolkit Implementation

We implemented our basic vocabulary in a parametric toolkit, using D3 data transformation (e.g., d3.nest), and visual presentation (e.g., d3.scale, d3.layout) functions. To facilitate writing these functions, we developed an application programming interface (API), through which we simply pass a set of parameters in the form of a JavaScript object (see Listing 1). The code is available at https://github.com/romsson/d3-gridding; it furthers the effort to serialize chart parameters [23,44]. This toolkit enabled us to easily explore various computation-based designs, using a parametric approach. We used it to generate all the remaining figures in this article, which are also available at a higher resolution in the code repository (e.g., https://romsson.github.io/d3-gridding/example/capture/display.html).

```
1 // Canvas properties
2 var width = 400, height = 300;
3
4 // Realistic data generation
5 var data = [1], nb_year = 4, nb_flow = 2,
6 nb_product = 10;
7 d3.range(nb_year).map(function(y) {
8 d3.range(nb_flow).map(function(f) {
9     d3.range(nb_product).map(function(p) {
10         data.push({"year": y, "flow": f, "product": p});
11     });
12 });
13 var nesting = ["year", "flow", "product"];
14
15 // Partition parameters
16 var params = [{
17     "size": function() { return [width, height]; },
18     "offset": function(d) { return [0, 0]; },
19     "mode": "vertical",
20     "valueHeight": "__agg_sum",
21     "orient": "up",
22     "padding": 2
23 },
24     {
25     "size": function() { return [width, height]; },
26     "offset": function(d) { return [0, 0]; },
27     "mode": "treemap"
28 ...}
```

Listing 1: Our toolkit parameters to generate a treemap bar chart

Fig. 15: Two different orientations (left figures) and orderings (right figures) of modules.

Fig. 16: Tables result from horizontal grids, with nested vertical grids.

**Tables** To create a table mock-up (□I \times □D) (Fig. 16), we first partition the chart space horizontally, according to the expected number of items in the data (□I). Each resulting *row module* is given the same height (similarly to a proportional segmentation in the graphic design perspective), and is then partitioned vertically, according to the expected number of dimensions (□D). The final *cell modules*
Stacks Typical stack layouts break the uniform vertical and horizontal separations (see Fig. 19) by compressing certain subdivisions, and changing their alignment. This way, a regular data table can have rows compressed on the left to reflect a cumulative perspective (using the stack operator) on each of the columns. This follows seminal work on Table Lens [39], which gives more space to rows or columns of interest, by squeezing in the others. If accumulation is performed on the first column of the table, it results in a vertical ranking of rows that sum up all the columns values, as done in Lineup [19] (Stack(D) × Values(D)) (Fig. 19, left). Changing the stacking baseline to an intermediate value (see Fig. 19, middle) can provide another form of comparison. Side-by-side comparisons [18] can also be done (see Fig. 19) with different stacking orientations to mock-up for example, a population pyramid, where groups are gender (Stack(I) × Bin(I) × Stack(D)) Fig. 19, right).

Bars To create a simple barchart mock-up (Values(D)) (Fig. 17), we first partition the chart space vertically, according to the expected number of items in the data (Stack(I)). We then nest a second vertical partition in each resulting bar module using the value data operator (Values(D)), which determines the height of each bar. This operation can be based on random values, or more realistic data. Horizontal barchart variations can be obtained by simply changing the partition pattern from Stack(I) to Group(I). Histogram variations (Bin(I) × Freq(I)) can be achieved by adding a bin operator to the first vertical segmentation (Bin(I)), and a frequency operator to the second (Freq(I)). Finally, grouped barchart variations (Group(I) × Stack(I) × Value(D)) (see Fig. 2) only require an extra initial vertical partition (or parent partition) using a group operator (Group(I)). Further variations can be explored by varying the width of each bar module (Fig. 17, center), for example to reflect non-equal bin sizes; or by nesting a grid partition in a histogram, to reflect the number of individual binned values (Grid(I)) (Fig. 17, right).

Grids Simple grids (Stack(I)) may, or may not be entirely filled with modules. For example, a 10 × 10 grid will be entirely filled. However, the same grid with |I| < 100 will not (see Fig. 18, left). We then use the orientation parameter to determine whether the empty space is located at the top of the grid, or at the bottom. The resulting cell modules can either nest other partitions, or determine placeholders for visual marks. For example, they can nest further grids (Grid(I) × Stack(I)), using either an individual local scale (Fig. 18, center), or a shared global scale (Fig. 18, right). By extension, they can also become small multiple chart variations [51] (Grid(I) × (chart)). Cell modules can also be used to place units, like Anthropographics [7] and other Isotypes [38], in a unit visualization [14]. Variations can be explored by repeating nested partitioning operations all the way down to the pixel level, for example to create mock-ups for VisDB-like visualizations [27].

Stacks (continued) Variations can be explored by recursively nesting partitions to create for example, Mosaic Plot mock-ups [20] (Stack(I) × Stack(D) × Stack(D) × (etc.)) (see Fig. 20). Mosaic plots suggest encoding dimensions at every recursion, alternating between horizontal, then vertical partitions (or vice-versa). This mechanism is somewhat similar to slicing and dicing a treemap layout [26] (I), although treemaps are generally better at space optimization, and they focus on encoding a single quantity per shape. To include more data, treemap partitions can be nested within each other (I × I), or in a vertical partition (I × I) to create for example, a treemap barchart.

Lines To create a linegraph mock-up (T × I × ξ(I)) (Fig. 21, left), we first need to partition the chart space vertically, according to the number of expected timesteps in the data (Stack(T)). We then partition each resulting timestep module according to the number of expected items (Stack(I)). The height of each of these nested modules is determined either by a proportion of the parent module’s (the timestep module) height, a random value, or more realistic data. We also assign each module a global scale, as the y scale at each timestep should be the same. Finally, we link each module. Slope graph [51] variations (Stack(I) × Filter(T) + ξ(I)) (Fig. 21, right) can be obtained by simply replacing the nested horizontal partition with a vertical partition, to which we apply a filter operator (Filter(T)), thus limiting the number of timesteps to two.
Lines (continued) Parallel coordinate variations are also very similar: they only require changing the data attribute of the initial vertical partition to the expected number of dimensions \(\text{(D,D)}\), and assigning the resulting nested modules local scales. Further variations can be explored by changing the partition patterns, for example to create semantic substrates \([47] ([\text{(D,D) \lor \text{Permutation(D,D)}]} \lor \text{ hist(T)})\), or treemap overlays \([16] ([\text{(D,D)} \lor \text{ Permutation(I,I)}])\) use the same layout, but their main components are proportions of the page content. Adjacency matrices \(\text{(\text{Permutation(I,I)})}\) use the same layout, but their main content is a permutation matrix—on items this time—and marginal bar charts encoding quantitative values, and not distribution (Fig. 26, right).

Coordinate systems Simple coordinate partitions \(\text{\text{\text{\text{(D,D)}}}}\) can be used to place modules according to \((x, y)\) values. These can nest other partitions, or determine placeholders for visual marks (e.g., in a scatterplot). Fig. 23 (left and center) typically illustrates nested partitions in a coordinate partition \(\text{(D,D) \lor \text{histogram})}\). Connected coordinate system variations \(\text{(D,D) \lor \text{hist(T)}}\) can be obtained by simply linking each module to show for example, the temporal evolution of items. By extension, we can easily create node-link di...
UpSet [33] \((D,D) \lor (\text{bar chart})\) is a recent technique to visually explore all possible set intersection permutations, and their quantification. Fig. 27 (right) shows UpSet’s main feature, which is the vertical matrix that is the permutation of all sets intersections. The matrix has a bar chart as header showing set distributions for each column; the horizontal bar charts show the quantities for each intersection. Additional bar charts are added to the rightmost part of the canvas to display more set properties.

As a known limit of UpSet is the vertical length of the matrix, we tested adding little vertical overlaps between grids, as our toolkit allows negative padding and margin values. Fig. 28 shows how bars can overlap slightly to create a bit of visual clutter, but acceptable as the trade off will be vertical compactness improvements. We also tested bar-piling, which is featured in small multiples [3] Fig. 28. More stacking strategies could be further explored as they are known to be an efficient way to pack more information on screen [13].

6 DISCUSSION AND DESIGN IMPLICATIONS

We have explored how iteratively subdividing the display space can help create visualization mock-ups at the graphical level, through a dialogue between graphic design, and parametric toolkit explorations. The graphic design perspective informed the viability of our display-to-data approach, and highlighted how the main operations needed to create mock-ups can be executed with little, or no data input. The parametric toolkit perspective illustrated how simple, iterative partitions of the display space, combined with light-weight data operations applied to expected data attributes (whether actually known or not), can help create a wide variety of more or less advanced visualization mock-ups.

We believe the main benefit of our display-to-data approach is that it allows designers to focus primarily on the visual form segment of the InfoVis pipeline (see Fig. 1). While our approach is not entirely data agnostic, it should prevent designers from having to worry about upstream data-related constraints. This suggests they should be able spend more time iterating over the output visualization—which can be shared with interlocutors—than on the input data in the early stages of the design process. For example, multiple parameters can easily be tweaked using our API to quickly assess how a visualization might fit on a screen [17]; whether partitions can be distinguished from each other; or whether a chart is likely to be overplotted. Therefore, it seems our approach could easily be integrated to broader visualization ideation methodologies, like Roberts et al.’s Five Design-Sheets [42], where our toolkit would provide a more principled way of exploring both data, and visual design spaces than traditional sketching. Indeed, design explorations can also go beyond the simple recreation of existing visualizations. Fig. 29 shows a naive attempt to nest all partition patterns within each other (e.g., \(M \times [3] \), \(M \times [3] \)), a method of benchmarking parameters that is in line with data-driven charts like scatterplot matrices, and other seminal work on spreadsheet structures [10, 57] that use grid-based layouts to visualize data using different charts, and different properties.

The main limit we see in our approach (and in mock-ups in general) is that it does not support high density charts, continuous charts (e.g., spark-line, streamgraph, curve links on graphs), and more complex data structures (e.g., hierarchical data, etc.). Also metadata (e.g., countries shapes), annotations, labelling, transitions, and interaction are not supported. We acknowledge that this currently makes it complicated to transition directly from the mock-up produced using our toolkit to final designs. That said, we argue that some marks and properties can be achieved by simply adding partition patterns to our toolkit, particularly non-standard patterns, like hexagonal and triangular patterns, and pie chart wedges; and by using other coordinate systems like polar coordinates. Some retinal properties like color could also be supported, although they do not comply with the look and feel of wireframes. An alternative would be to use textures [4], as they are a reasonable alternative to color, and can be generated using further partitions patterns [56]. Other graphic design elements could also be added to the mock-ups (see Figures in section 3), such as reference structures [30] (like map graticules), as they can facilitate both construction, and comparison.

However, adding more expressiveness to mock-ups could also add visual complexity, as well as API complexity. This could make our toolkit less usable, and could impair the creative process.

7 CONCLUSION AND PERSPECTIVES

In this article, we have presented a principled design approach for creating visualization mock-ups from the graphical level downwards. It relies mainly on the iterative subdivision of the display space, and the progressive inclusion of data. It is highly modular, and can help rapidly iterate over design ideas for both basic charts, and more advanced visualization interfaces. We hope this work will motivate and inform the development of future high-level prototyping tools for InfoVis.

Nevertheless, we acknowledge our approach is bound to a number of constraints (e.g., grid systems from the graphic design perspective, and partition patterns from the parametric toolkit perspective), which may be limiting. Therefore, we would like conclude by echoing the words of famous graphic designer Muller-Brockmann, and of famous architect Le Corbusier. The former stated that the “grid system is an aid, not a guarantee. It permits a number of possible uses, and each designer can look for a solution appropriate to his personal style. But one must learn how to use the grid; it is an art that requires practice.” While the later mentioned “I still reserve the right, at any time, to doubt the solutions furnished by the Modulor”, keeping intact my freedom, which must depend on my feeling rather than my reason” (in [22]).

REFERENCES


3 Le Corbusier’s anthropomorphic scale system for architecture.