F QK<"320 3333 lei h0345; 3
Eurographics Conference on Visualization (EuroVis) 2014
H. Carr, P. Rheingans, and H. Schumann
(Guest Editors)

Volume 33 (2014), Number 3

Lyra: An Interactive Visualization Design Environment

Arvind Satyanarayan[1] and Jeffrey Heer[2]

[1] Stanford University [2] University of Washington

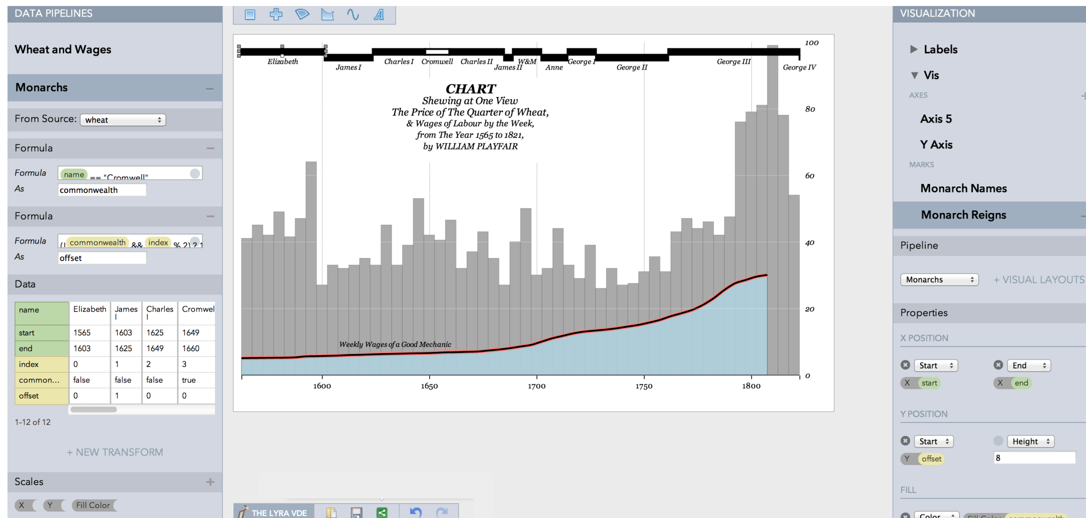


Figure 1: *The Lyra visualization design environment, here used to recreate William Playfair's classic chart comparing the price of wheat and wages in England. Lyra enables the design of custom visualizations without writing code.*

Abstract

We present Lyra*, an interactive environment for designing customized visualizations without writing code. Using drag-and-drop interactions, designers can bind data to the properties of graphical marks to author expressive visualization designs. Marks can be moved, rotated and resized using handles; relatively positioned using connectors; and parameterized by data fields using property drop zones. Lyra also provides a data pipeline interface for iterative, visual specification of data transformations and layout algorithms. Visualizations created with Lyra are represented as specifications in* Vega*, a declarative visualization grammar that enables sharing and reuse. We evaluate Lyra's expressivity and accessibility through diverse examples and studies with journalists and visualization designers. We find that Lyra enables users to rapidly develop customized visualizations, covering a design space comparable to existing programming-based tools.*

Categories and Subject Descriptors (according to ACM CCS): H.5.2 [Information Interfaces]: User Interfaces—GUI

1. Introduction

When creating custom visualizations, designers must contend with a variety of concerns, including perceptual effectiveness, audience familiarity and aesthetic choices. In addition to data values, a designer may seek to convey the semantics and connotations of the data and the context of data collection. Judiciously designed visualizations can foster effective communication and engage interest.

Consider *U.S. Gun Deaths in 2013* [Per13] from the firm Periscopic (Figure 2). Thin orange lines trace across a black background in an animated arc, depicting the span of people's lives. They abruptly halt, and dots drop to a baseline, to symbolize lives lost to gun violence. The arcs resume their trajectories, colored grey to depict the remaining years the victims might have lived. While a bar or line chart could communicate these statistics, the unique design decisions seek to convey the data in tandem with the emotional weight of this lost potential. As the designers note, *"What the [dataset] does not contain is an assessment of the potential life that was stolen from these individuals"* [Per13].

Designing visualizations of this caliber is challenging, and current design tools lie along a spectrum of expressiveness.

© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

delivered by EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

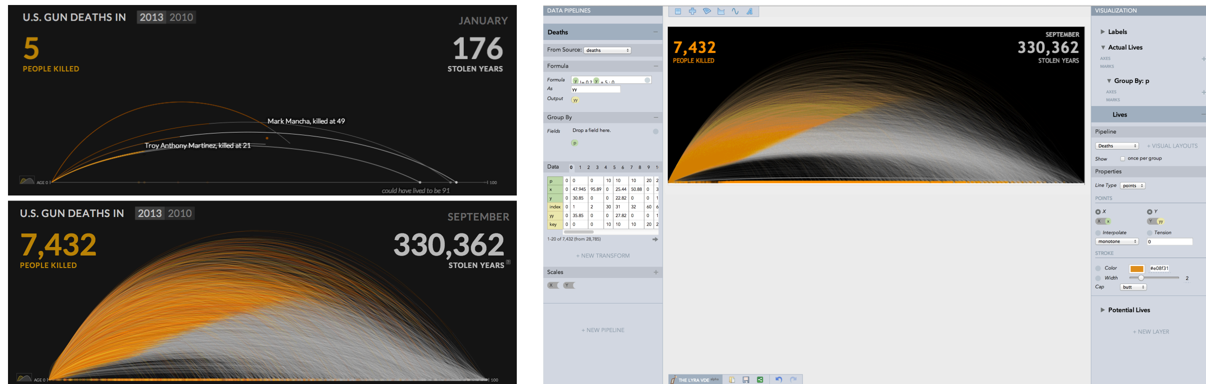


Figure 2: U.S. Gun Deaths in 2013*, a visualization by Periscopic (left). The use of arcs and color reflects the emotional weight of its subject: the devastating effect of gun violence. A static version of the visualization recreated in Lyra (right).*

At one end are *chart typologies* [Wil05]: pre-defined palettes of chart types (bar charts, treemaps, etc.) that make numerous design decisions on behalf of the user. By trading expressiveness for efficiency, chart typologies inhibit the creation of novel visualization designs. At the other end are vector drawing tools [Tuf] and programming [Pro, BOH11]. Most drawing tools do not provide data-driven abstractions, resulting in time-consuming and error-prone work. Programming imposes a wide gulf of execution [HHN85] and a poor "closeness of mapping" [GP96] between language syntax and graphical output. Some visualization systems [BH09, BOH11, Wic09] reduce this gulf with grammars that describe visual primitives. However, using text to express visuals incurs a significant articulatory distance [HHN85].

We present *Lyra*, an interactive design environment for custom visualization that is comparable in expressiveness to programming-based tools. With Lyra, designers add graphical *marks* to a canvas and associate data fields with mark properties. Visual *data pipelines* enable data transformation and advanced layout algorithms. Lyra incorporates familiar interactions found in drawing and diagramming tools: *Handles* can be used to interactively move, rotate, and resize marks; *connectors* relatively position marks; and *drop zones* allow data binding via drag-and-drop. By exposing all mark properties as data binding targets, Lyra provides the fine-grained control needed to produce unique visualizations. Direct manipulation of marks further reduces the articulatory distance for design [GP96, HHN85].

Informed by prior work such as the Grammar of Graphics [Wil05, Wic09], Protovis [BH09, HB10] and D3 [BOH11], Lyra represents visualizations using the *Vega* visualization grammar [Veg13]. Users can export their designs as images or as reusable components in Vega's JavaScript Object Notation (JSON) format. As a result, Lyra visualizations can easily be published on the web, shared and repurposed. In short, Lyra contributes a novel interactive system for visual specification of grammar-driven visualizations.

We focus on the problem of crafting custom visual encodings. Static visualizations continue to be widely used across online and print media. Though important, we leave specification of related interaction techniques (e.g., brushing & linking) to future work. The current open-source Lyra system provides a platform for further design tool research.

To evaluate Lyra's expressiveness, we demonstrate its use to create diverse example graphics that are difficult or impossible to construct with existing interactive tools. To evaluate accessibility, we conducted a first-use study with designers and journalists. Participants described Lyra's interaction model as *"natural"* and *"intuitive"*, and successfully used Lyra to rapidly prototype custom visualizations.

2. Related Work

Lyra builds on visualization systems research spanning chart typologies, toolkits, grammars and graphical design tools.

2.1. Chart Typologies

Chart templates, as found in spreadsheets and online services (e.g., Many Eyes [VWvH*07], Google Fusion Tables), are a common means of creating visualizations. Such *chart typologies* [Wil05] involve selection rather than creation, requiring a few clicks to produce a chart. However, users are restricted to the available chart types and may only customize a small number of parameters. Lyra supports a more expressive range of visualization designs, without using a textual programming language. Moreover, Lyra exports visualization components that can extend existing typologies.

2.2. Visualization Toolkits

To enable custom visualizations, researchers have developed a variety of programming toolkits [LAC*92, Fek04, Wea04, HCL05, Fla13]. Some [Fek04, Wea04] provide a class hierarchy of visualization widgets, where new visualizations are created by subclassing existing components or writing new ones. Others [Fla13, HCL05, LAC*92] create visualizations

© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and John Wiley & Sons Ltd.

using composable operators for data transformation, layout and encoding. The operators allow fine-grained control over visual output and are amenable to graphical data flow specifications [LAC*92]. However, novel designs often require programmers to create new operators [HB10], limiting these tools to software engineers. Lyra is intended to make custom visualization design accessible to a less technical audience.

2.3. Visualization Grammars

In *The Grammar of Graphics* [Wil05], Wilkinson argues we should "shun chart typologies" and introduces a grammar for specifying a wide range of data graphics. His grammar proposes a set of primitives for statistical graphics: data and scale transforms; visual elements with "aesthetic" attributes; and guides such as legends and axes. Wickham's ggplot2 [Wic09, Wic10] follows this design, and a similar formulation underlies Tableau [STH02]. By formalizing a combinatorial design space, these tools are more expressive than chart typologies. However, they are designed to support rapid exploratory analysis, not custom visualization design. These systems make many default decisions on behalf of the user and limit one's control over visual design parameters.

In contrast, Protovis [BH09] and D3 [BOH11] are grammar-based systems in which the properties of graphical *marks* such as bars, lines, arcs, and text labels are parameterized as functions of an underlying data set. Any number of mark instances, backed by one or more data sets, are composed to form a graphic. Protovis [BH09, HB10] demonstrates how a lexicon of basic mark types can support a diversity of visualization designs. D3 (Data-Driven Documents) [BOH11] applies a similar model directly to elements of a web page's Document Object Model. Data elements are associated with graphical objects via a relational join operation; as data changes objects may enter, exit, or update in the display [HB10, BOH11]. Custom property definitions can be defined and animated across each of these phases. However, both of these tools require JavaScript programming.

2.4. Interactive Visualization Design Tools

Lyra appropriates common direct manipulation techniques found in vector-based drawing tools such as Adobe Illustrator. These systems, often used to produce static graphics, offer flexibility and a close articulatory distance [HHN85]. However, they do not provide visualization-specific abstractions, resulting in a time-consuming and error-prone process.

Recently, Bret Victor demonstrated a data-driven drawing tool [Vic13] that combines geometric constraints with *imperative* procedures over data. Alongside an interactive canvas and data table viewer, the tool includes programming structures with lexical scope and control flow via loops and conditionals. To form a basic bar chart, designers must define loops to create and position each bar. The tool supports purely geometric constructions, rendering some layouts inexpressible. Lyra combines manipulation of marks with a *declarative* approach to design [HB10]. When a designer associates a mark with a data set, Lyra instantiates a mark instance for each datum; control flows such as loops are implicit. More advanced algorithms, such as layout routines, are accessible as part of a graphical data pipeline.

Lyra's use of drag-and-drop to associate data fields with mark properties is inspired by Tableau and its predecessor Polaris [STH02]. In Tableau, a "schema" panel lists available data fields which can be dragged to "shelves" on the periphery of the visualization to specify data groupings and visual encodings. Lyra extends this mode of interaction into the visualization canvas. Property *drop zones* overlay mark instances during drag operations, providing a more direct data-mapping target. Lyra also provides property inspectors akin to Tableau's shelves, but which expose more fine-grained details to enable custom design. Whereas Tableau is optimized for efficient construction of analytic graphics, Lyra is optimized for expressive design control.

Roth et al.'s SageBrush [RKMG94] supports similar interactions. Users can drag-and-drop "partial prototypes" for spatial encodings and "grapheme" (mark) primitives such as lines and bars. Custom grapheme properties are manually selected via menus, then exposed as drop-target icons. Lyra refines the SageBrush model in several ways. Partial prototypes in SageBrush implicitly define scale transforms and layout; Lyra's grammar decouples these to support richer designs. Lyra's data pipelines provide an extensible set of data transformations; different marks can be driven by unique pipelines and composed. Lyra eschews iconic abstractions in favor of direct drop zones for mark properties, without need of explicit enumeration via menus. In summary, Lyra is designed to support a larger expressive space and more fine-grained control than either SageBrush or Tableau.

3. The Design of Lyra

Lyra was developed through an iterative user-centered design process. We held formative interviews with representative users, such as visualization designers and journalists, to understand their design process and the limitations of their existing tools. These users evaluated low-fidelity prototypes and later interactive prototypes. In this section, we describe the constituent components of the Lyra system, and justify our design decisions based on the results of our evaluations.

3.1. The Vega Visualization Grammar

Lyra is built atop the Vega [Veg13] visualization grammar, which provides a set of basic abstractions necessary for constructing visualizations. These abstractions are in turn drawn from *The Grammar of Graphics* [Wil05] and prior visualization toolkits including Protovis [BH09] and D3 [BOH11]. Lyra uses the Vega grammar as both an internal model and

© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and John Wiley & Sons Ltd.

external file format. Like Protovis and D3, Vega uses data joins and graphical marks to enable a breadth of designs. Unlike these systems, Vega uses a declarative JSON format and defines reusable chart components. Lyra visualizations can be exported as Vega JSON files to drive web-based visualizations, or as static PNG or SVG images.

Data. Lyra assumes a tabular data model: a collection of records with named attributes of a given data type. A data tables may also be organized into a hierarchy using an appropriate data transformation, e.g., a *group by* transform groups records with matching values for a specified `key` field.

Data Transforms. Transforms manipulate data prior to visualization, and include statistical (e.g., filtering, grouping) and visual encoding (e.g., cartographic projection, treemap layout) routines. Transforms produce data sets as output and may accept multiple parameters (often data fields).

Scales. Scales are functions that map data values to visual properties such as position, shape, and color. Lyra includes both ordinal and quantitative (e.g., linear, log, quantile) scales, along with common shape and color palettes.

Guides. Guides visualize scales, providing reference marks to aid interpretation. *Axes* visualize scales over a spatial domain, and include ticks, labels, and optional grid lines. *Legends* visualize scales for color, shape or size encodings.

Marks. Marks are geometric shapes with named visual properties. Property values can be set manually or bound to data. A single mark definition can be bound to only one data set; Lyra instantiates one mark instance per data record. Akin to Protovis [BH09, HB10], the available marks in Lyra are: *rectangle*, *arc*, *area*, *line* (including closed polygons), plotting *symbols*, and *text* labels. All marks share common properties (e.g., `x`, `y`, `fill`, and `stroke`) but may also have unique properties (e.g., `angle` and `radius` for *arcs*).

3.2. The Lyra Interface

The Lyra interface, as shown in Fig. 1, is split into three sections. The left-hand panel (Fig. 3a) depicts *data pipelines*: chains of data transformations applied to a data source. A pipeline's inspector provides a paginated data table showing the output of the pipeline, buttons to add new transformations, and a list of scales defined over data fields. The right-hand panel (Fig. 3c) contains inspectors for graphical elements such as marks, axes, and legends. These elements are grouped into *layers* to determine coordinate spaces and z-ordering. These inspectors list all visual properties (position, fill color, angle, etc.) along with widgets to manipulate them. The central panel contains the visualization canvas where graphical elements may be directly manipulated.

3.3. Data Pipelines

Lyra's left-hand panel contains *data pipelines*: workflows of transforms applied to input data. Clicking a pipeline reveals

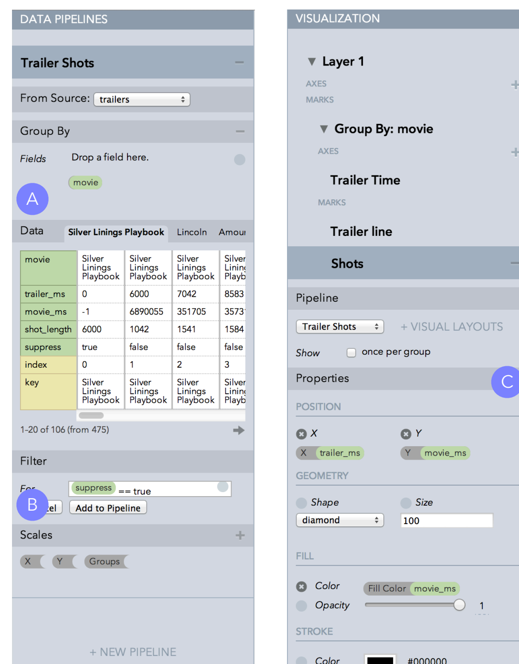


Figure 3: *Lyra's side-panels for data pipelines (left), and visual properties (right). (a) Data table showing the current output of the pipeline; (b) Scale transforms defined over fields in the pipeline. (c) A property inspector for a symbol mark type; two properties have been mapped to data fields.*

an inspector that lists applicable transforms and presents a paginated table view of transformed data.

Data Table View. Data pipelines include a data table view, using a layout inspired by Bret Victor [Vic13]. The first column in the table view lists field names, enabling vertical scanning. Subsequent columns display individual records (Fig. 3a). Field names in the first column are interactive: clicking a field sorts the table by that dimension, drag-and-drop can be used to bind fields to mark properties. Fields are colored by their source: green for fields in the original data and yellow for fields derived by a transform. For example, *formula* transforms adds new fields based on mathematical expressions. When *group by* transforms are applied, one tab for each group appears above the table view.

Authoring Transforms. Users can add a transform by clicking the corresponding icon and configuring its parameters. Users may preview the effect of applying a transformation in a popover. Once a transformation is added to the pipeline, adjusting its properties is reflected in real-time across the table view and the visualization.

Scales. The inspector also lists all scales defined over data fields in the pipeline (Fig. 3b). Lyra automatically instantiates scales when a field is associated with a mark property. The scale domain is defined over the field values; the range is determined using production rules described below. Users

© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and John Wiley & Sons Ltd.

can also create scales manually. Users can drag scales onto mark properties to apply a scale transform, or click a scale to access an editor dialog (see Fig. 5e). When editing a scale that is not represented by an axis or legend, a transient guide is shown in the canvas to convey the effect of scale changes.

Rationale. Our initial prototypes hid raw data values in favor of exposing only the table schema. However, user evaluations indicated this was insufficient. Users noted that it was difficult to determine the effect of a data transformation based only on the visualization. The incremental nature of visualization design can lead to unexpected intermediate output, for example setting the `height` of a rectangle mark can cause all mark instances to overlap if no `x` or `width` property has been set. Later prototypes introduced a full data table view, to enable inspection of raw values and expose the current data organization.

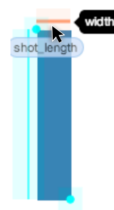
Similarly, early prototypes masked the presence of scales: mapping data to visual properties automatically instantiated a scale, but they were not explicitly exposed in the interface. When users attempted to construct visualizations, we found that this significantly restricted their expressiveness. For example, it is often necessary to specify custom ranges for scales rather than rely on preset ranges. Such modification is difficult to do without surfacing scales as a first-class construct. Later evaluations found that users additionally had trouble identifying the purpose of scales, or the effects of scale modification, if the scales were not explicitly represented on the visualization by an axis or legend guide. In response, we introduced transient guides.

3.4. Composing Visual Elements

Visualizations in Lyra are compositions of visual elements: graphical *marks* and *guides*. Elements are grouped together into *layers*, which define local coordinates and establish z-ordering. Lyra's right-hand panel lists the layers and their elements (Fig. 3c). Elements are added to a visualization by creating them within this panel or by dragging a mark from the mark palette. When an element is selected, an inspector presents all the element's associated properties. Property values may be edited directly or set via drag-and-drop of data fields with changes reflected on the visualization in real-time. Hovering over a property displays a guide overlaid on the visualization to illustrate how that particular property affects the rendered output. Visual elements can also be manipulated directly on the visualization canvas.

Handles in the canvas area can be used to interactively move, rotate and resize selected elements. A mark definition will typically render one mark instance per datum in the visualization. To reduce visual clutter, selecting a mark displays handles only on the instance that was clicked. However, when a user adjusts the handles, the change is reflected simultaneously across all mark instances.

Connectors. Marks can be positioned relative to one another using diamond-shaped connectors. Dragging a target mark onto a host mark's connector establishes a connection: the target mark's position is now determined by the host's properties. Changes to the host mark automatically propagate to all connected targets. Connectors are particularly useful for positioning text labels relative to other marks.



Drop zones. Lyra uses drop zones to associate data fields with mark properties. When dragging a data field, drop zones are overlaid on the visualization canvas. Each drop zone comprises a shaded region and a guide line or point to indicate the corresponding mark property (e.g., `x`, `width`, etc.). Hovering on a drop zone highlights it and shows the property name in a tooltip. Dropping a field then establishes a mapping between the data field and the mark property. To avoid clutter, Lyra shows drop zones only for the currently selected item. When dragging a data field, users can hover and pause over a mark instance to make it the selected item.

Rationale. Surfacing all properties in the inspector was an immediate first step to ensure that Lyra maintained Vega's expressivity. Users noted that these inspectors were akin to Tableau's "shelves," a familiar interaction paradigm for many of them. However, there remained a clear opportunity to further narrow the gulf of execution [HHN85] by pushing interactions to the visualization canvas itself. For example, we observed users attempting to select, move, or resize marks currently visualized on the canvas.

As users cited familiarity with drawing tools, we sought to reuse familiar interaction mechanisms with *handles* and *connectors*. However, there is not a similarly established interaction mechanism for data-property bindings. We ultimately arrived at our *drop zones* design by prototyping a number of alternatives. One such alternative incorporated flow menus [GW00]. When dragging a data field over a mark on the canvas, a flow menu would appear listing all mappable visual properties. When dragging the field over a property, a submenu would appear listing appropriate scale types given the type of the data field, and the particular property. For example, for fields with numeric data, this submenu offered all quantitative scale types including linear, logarithmic, and so forth. Dropping the field over a particular scale type established a mapping and instantiated the appropriate scale.

In addition to testing designs with users, we analyzed them using the Cognitive Dimensions of Notation heuristics [GP96]. Data mapping through flow menus, for example, provided a *visible* and *consistent* interface—regardless of the mark type, all properties were consistently ordered within the top-level menu. Although exposing scale types in the submenu arguably reduced *error- proneness* (as Lyra

© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and John Wiley & Sons Ltd.

need not infer a scale type), it increased the *diffuseness* (or verbosity) of the interface. User feedback also revealed that selecting an option from this submenu was a *hard mental operation* as it forced them to select a particular scale type up front. Many users perceived this as a *premature commitment*. Perhaps most troublesome, given our goal of reducing the gulf of execution, was the lack of a *closeness of mapping*: properties were listed as menu items, one after another.

Drop zones, on the other hand, achieve a high *closeness of mapping* as they overlay the canvas in a way that corresponds to the property they represent. For example, a rectangle's `x2` drop zone is shown extending from the left edge of the canvas to the right-most edge of the rectangle. Dropping a field over a drop zone performs *scale inference* (described below) to reuse an existing scale definition or instantiate a new one. Although this may increase *error-proneness*, it decreases *diffuseness* and reduces the *hard mental operations* flow menus presented. One limitation of drop zones is a subtle lack of *consistency*; for example, a tall rectangle mark will present a larger `height` drop zone than a shorter one. We mitigate this issue by showing drop zones only for the currently selected mark.

3.5. Scale Inference and Production Rules

When a user binds a data field to a mark property, Lyra performs *scale inference* in an attempt to reuse existing scale definitions. Lyra searches for an existing scale with the field as its domain. If a scale is found, it is reused if its range type is appropriate (e.g., spatial or color values). If no scale is found or the range type does not match, Lyra instantiates a new scale: ordinal for categorical data or linear for quantitative data, along with a default range based on the property type (e.g. `width` for `x` properties).

To accelerate common encoding decisions, Lyra also uses a set of context- and mark-specific *production rules* to determine intelligent defaults. These production rules may set additional properties of the mark or add new graphical elements to the canvas. For example, dropping a field over a rectangle mark's `width` drop zone automatically binds the `x` property as well to correctly position each rectangle. Dropping a field over a spatial property may add an axis; dropping a field over a color property may add a legend.

Rationale. Scale inference and production rules were informed primarily by early user feedback. Without these features, users had to manually create every aspect of the visualization, which they found to be tedious. Users did not expect to have to specify a scale definition on every data mapping operation, and expected axes or legends to be automatically added as appropriate. We found that the features did alleviate this tedium but, interestingly, users subsequently requested a method of circumventing them *"if they knew better"*. As a result, although Lyra performs scale inference on every data field mapping, production rules are only evaluated if the data field is dropped over a drop zone. Users may sidestep the rules by working directly with property inspector instead. We fully enumerate Lyra's scale inference procedure and production rules in supplementary material.

4. Evaluation: Example Lyra Visualizations

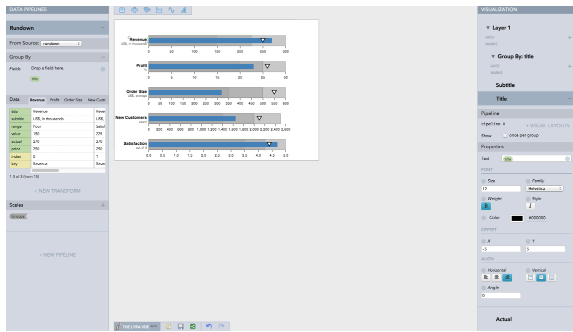
One of Lyra's primary goals is to enable an expressive design space. With Lyra, it should be possible to create visualizations that would have previously required programming. To assess the extent to which this goal has been met, we constructed a diverse collection of example visualizations, including those shown in Fig. 4. These examples compose multiple mark types, and many require multiple data pipelines. For example, Fig. 4(d) uses line, symbol, and text marks to convey two datasets: train routes and stations. Fig. 4(h) demonstrates that Lyra's integration of data pipelines and graphical manipulation is necessary to maintain expressiveness: shading the bars requires a data pipeline with *Group By* and *Formula* data transformations applied.

Similarly, exposing visual layout inspectors allows for rapid design iteration. In Fig. 4(c), for example, the *force-directed layout* inspector exposes parameters such as link distance, strength and gravity; adjusting them re-renders the layout in real-time. The layout also augments direct manipulation on the canvas: designers can brush to select nodes and double-click to pin them. Together, these facilitate a converging process: pinning satisfactory nodes, adjusting layout properties, and re-running the layout to reposition unpinned nodes. This process would be cumbersome using only D3's force-directed layout [BOH11]: after programming the layout, adjusting parameters requires editing the code and refreshing the browser. Pinning nodes then requires inspecting the properties of each rendered node individually and copying the `x` and `y` positions into the raw dataset.

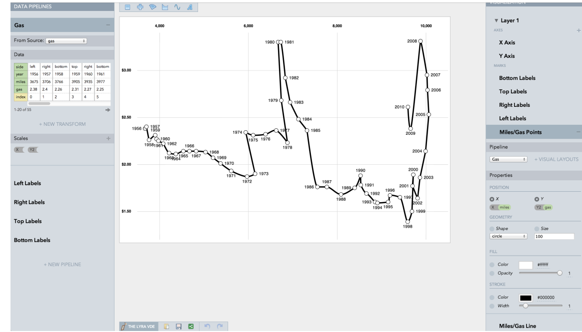
Figure 5 illustrates how Lyra can be used to compose *Dissecting a Trailer* [CCB13], a New York Times visualization that uses over 350 lines of custom JavaScript and D3 [BOH11] code. Marks are added to the visualization by dragging them from the palette and dropping them onto the canvas (Fig. 5a). Their properties can be bound to data by dragging fields from a pipeline's data table and dropping them over corresponding *drop zones* on the canvas (Fig. 5b). Using appropriate inspectors, we can add data transformations (Fig. 5c-d) and specify mark properties (Fig. 5e).

Limitations. The visualizations in Figures 4 and 5 demonstrate that Lyra enables an expressive design space, but creating these examples also reveals some limitations. Vega currently lacks support for polar coordinates. As a result, Lyra cannot (yet) provide *arc* mark connectors or produce radial axes, making it difficult to recreate classic visualizations such as Nightingale's Rose or Burtin's antibiotics chart. Additionally, Lyra only supports the RGB color space, while Vega also supports HSL, LAB, and HCL. These color spaces can facilitate perceptually-sound designs. We plan to address these limitations in future iterations of Vega and Lyra.

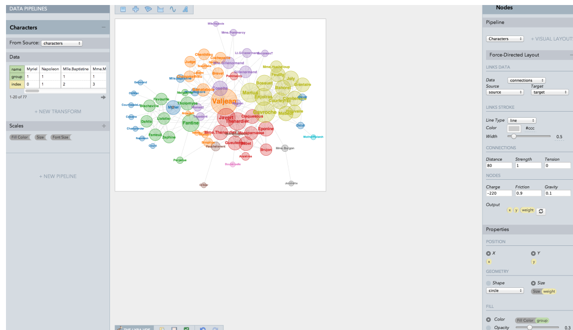
© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and John Wiley & Sons Ltd.



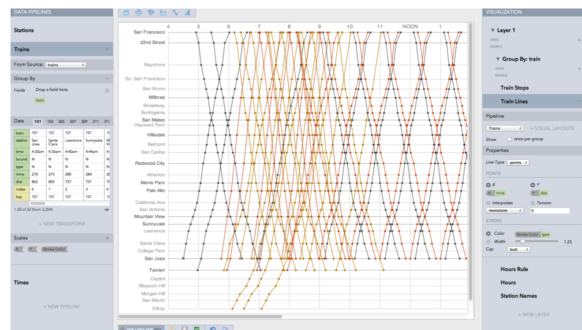
(a) Bullet chart using rectangle and symbol marks grouped by category. Labels are positioned via a left-edge connector on rectangles.



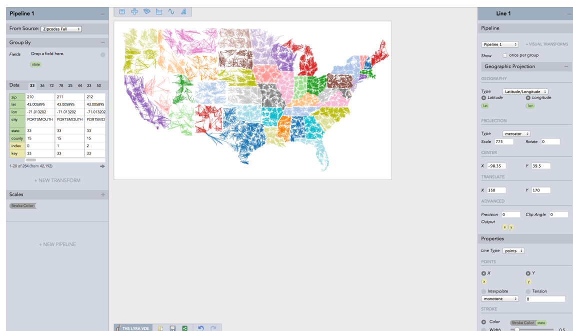
(b) A recreation of *Driving Shifts Into Reverse* from The New York Times, originally published May 2, 2010.



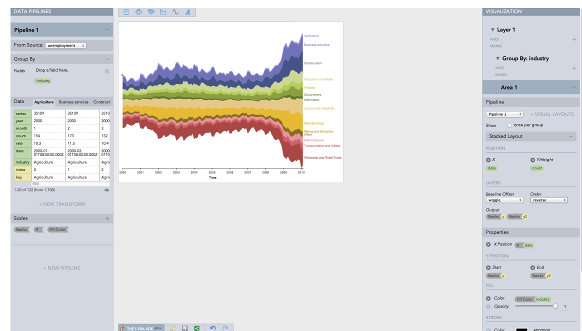
(c) Character co-occurrences in Les Misérables. Colors represent cluster memberships computed by a community-detection algorithm.



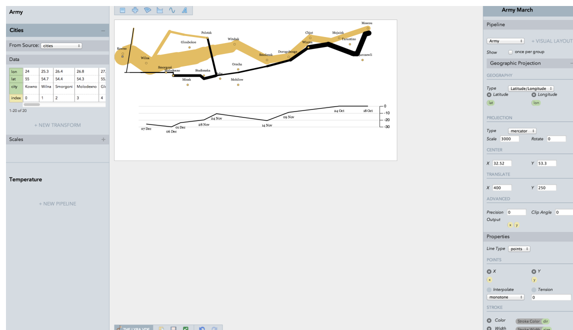
(d) The schedule of the San Francisco Bay Area's CalTrain service in the style of E. J. Marey's Paris train schedule.



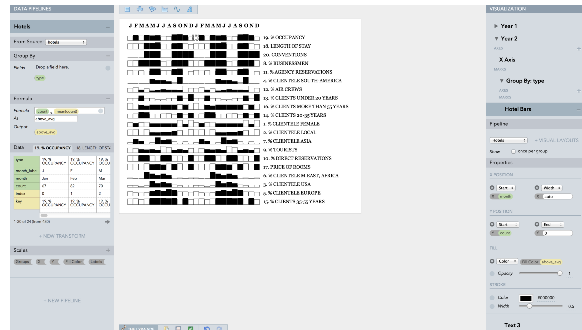
(e) ZipScribble by Kosara [Kos06]. A *geo* layout encoder is used with line marks to connect latitude and longitudes of zip codes.



(f) A streamgraph of unemployed U.S. workers by industry, using a *stack* layout with a `wiggle` [BW08] offset.



(g) Minard's map of Napoleon's Russian campaign. A *geo* transform encodes spatial positions; army size maps to line stroke width.



(h) Jacques Bertin's analysis of hotel patterns. *Group by* and *formula* transforms are used to shade bars with values above the mean.

Figure 4: *Example visualizations demonstrating Lyra's expressivity.*

© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and John Wiley & Sons Ltd.

Figure 5: *Using Lyra to recreate the New York Times' Dissecting a Trailer. (a) Drag a line* mark *onto the canvas. (b) Drag a field from a* pipeline*'s data table to a* drop zone *to map it to a mark property. (c) Add a "group by"* data transform *to create a hierarchy. (d) Edit a* scale *definition to reverse the range. (e) Use a* connector *to anchor text marks to the rectangles.*

5. Evaluation: First-Use Study

Lyra was designed to support both expressive and *accessible* visualization design: users should not require coding expertise to be able to construct custom visualizations. To evaluate Lyra's accessibility, we conducted first-use studies with 15 representative users including 6 data analysts / visualization designers, 5 data journalists, and 4 graduate students in data visualization. The median self-reported visualization design expertise was 7 on a 10-point scale, while programming expertise ranged between 2–8 on a 10-point scale. These users all use visualization as a communicative medium but their processes for creating them vary. The visualization designers and grad students were more technically proficient and typically use D3, whereas the data journalists rely on chart typologies (Microsoft Excel) or grammar-based systems (Tableau) that do not require programming. Some journalists also reported eschewing visualization systems in favor of drawing programs such as Adobe Illustrator.

Methods. We began each study with a 10 minute tutorial. We then asked participants to design three graphics: a bar chart of medal count by country at the 2012 Olympics (T1), a grouped or stacked bar chart of medal counts by medal type and country (T2), and a trellis plot of barley yields (T3, Fig. 6). These tasks were designed to ensure participants interacted with all aspects of Lyra. Each task was more difficult than the previous, intending to first familiarize participants with the Lyra design process, and then challenge them. Participants were encouraged to think-aloud and were de-briefed at the end. Sessions lasted approximately 45 minutes, after which we administered a post-study survey.

Successes. Users quickly learned Lyra's interaction model and all users, regardless of their technical expertise, successfully completed all three tasks with minimal guidance (100% task completion rate). Users completed the first two tasks in just a few minutes, the more complex third task took longer (T1: median time = 1:33, inter-quartile range (IQR) = 0:51; T2: median = 2:43, IQR = 2:57; T3: median = 10:24, IQR = 4:00). In a post- study survey, users rated Lyra's interface highly: drop zones felt natural to use ($\mu = 4.4$, $\sigma = 0.57$ on a 5-point Likert scale), connectors helped to relatively position marks ($\mu = 4.3$, $\sigma = 0.49$), and a pipeline's data table helped evaluate context ($\mu = 4.4$, $\sigma = 0.51$). Handles were found useful for resizing and positioning ($\mu = 3.8$, $\sigma = 0.45$) but users noted that the properties they control are typically mapped to data. When asked to recount their experience, users described drop zones as *"natural"* and *"intuitive."* One user stated, "*it's like* literally *saying 'put that there.'*" Others drew comparisons to Tableau's shelves: *"[shelves] don't always behave like I expect them to but [drop zones] make me feel*

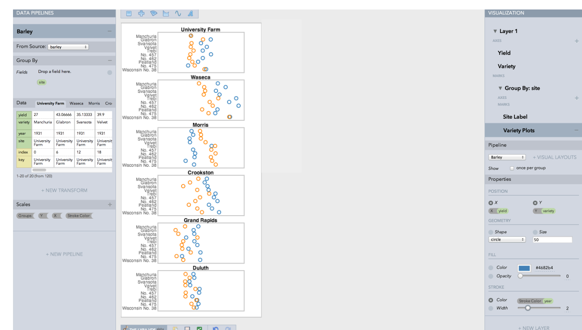


Figure 6: *Users were asked to recreate a version of the barley yields Trellis display by Becker et al. [BCS96]*

© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and John Wiley & Sons Ltd.

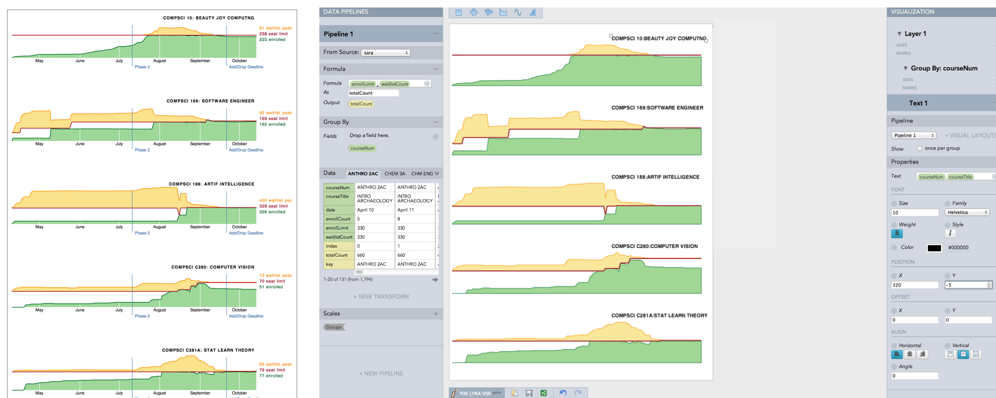


Figure 7: *A user approximately recreates a D3 visualization (left, requiring 4-6 hours) in Lyra (right, requiring 10 minutes).*

more in control.” One participant ended his session by saying that *“there’s a real joy in using Lyra.”*

Two journalists who participated lead data visualization teams in their organizations. They appreciated that Lyra took cues from familiar drawing tools. They welcomed Lyra’s image export options, particularly SVG export, as the visualizations they produce are often reutilized in print media. One suggested that Lyra could be a powerful training tool that could help familiarize his team with the process of designing visualizations from the ground-up.

Shortcomings. We also observed that Lyra posed certain challenges for our participants. Although users found drop zones natural and intuitive, they noted problems with the current implementation. First, when users missed a drop zone by a few pixels, they expected Lyra to infer their intent. Second, when users successfully dropped a field, they would lose track of the currently selected mark if it was repositioned. A third shortcoming was Lyra’s lack of support for undo, which led users to become more hesitant to freely explore. Undo support has since been added to Lyra, and we plan to address the remaining issues in future versions. For example, increasing the activation area for a drop zone could help to address the first issue. Staggered animated transitions could help users better track changes [HR07] to marks.

Finally, several users mentioned that learning from and repurposing existing visualizations is an important part of their design process. They found the blank canvas to be an intimidating starting point. We anticipate that providing a gallery of examples (including those in this paper) that users can import, reuse and modify could help mitigate this issue.

6. Conclusion and Future Work

In this paper we contribute Lyra, a direct manipulation environment for visualization design. By reducing both tedium and required technical expertise, Lyra makes custom visualization design more accessible to a broader audience. Direct manipulation techniques such as handles, connectors, and drop zones reduce the gulf of execution, while Lyra’s data pipeline and visual canvas help bridge the gulf of evaluation. A diverse collection of examples demonstrates Lyra’s expressiveness, including many designs that are not expressible in current interactive visualization tools. Representative users are able to create custom visualizations much more quickly than with current tools, and report that they find Lyra’s interface *“natural”* and *“intuitive.”*

There are a number of directions for future work. For example, Lyra uses direct manipulation techniques for composing graphical marks. How can these techniques be extended to support other tasks? Although Lyra’s primary focus is as a design tool, data visualization inevitably requires data cleaning and transformation. Lyra’s data pipelines offer sufficient flexibility to support analytics tasks, but requires familiarity with pipeline operators. What direct manipulation techniques could be further incorporated to specify complex data transformations (c.f., [KPHH11])?

An exciting challenge is the design of *interactive* visualizations without recourse to programming. A straightforward approach is to include common interactions, similar to existing toolkits [Fla13, HCL05]. As with chart typologies, such “interactor typologies” can enable rapid development of common interactions, but may restrict the design of novel interaction schemes. A deeper question concerns the design of combinatorial interaction primitives that complement the graphical primitives provided by visualization grammars.

Lyra is available as open-source software at `http://idl.cs.washington.edu/projects/lyra/`.

Acknowledgements

We thank our study participants for their helpful insights and for accommodating us in their deadline-driven schedules. We also thank Kanit “Ham” Wongsuphasawat, Erica Savig, Katherine Breeden, Jerry Talton, and Kathryn Papadopoulos for their feedback. This work was supported by an SAP Stanford Graduate Fellowship and DARPA XDATA.

© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and John Wiley & Sons Ltd.

References

[BCS96] BECKER R. A., CLEVELAND W. S., SHYU M.-J.: The visual design and control of trellis display. *Journal of Comp. & Graphical Statistics 5*, 2 (1996), pp. 123–155. URL: http://www.jstor.org/stable/1390777. 8

[BH09] BOSTOCK M., HEER J.: Protovis: A graphical toolkit for visualization. *IEEE Trans. Visualization & Comp. Graphics 15*, 6 (2009), 1121–1128. 2, 3, 4

[BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D3: Data-Driven Documents. *IEEE Trans. Visualization & Comp. Graphics 17*, 12 (2011), 2301–2309. 2, 3, 4, 6

[BW08] BYRON L., WATTENBERG M.: Stacked graphs–geometry & aesthetics. *IEEE Trans. Visualization & Comp. Graphics 14*, 6 (2008), 1245–1252. 7

[CCB13] CARTER S., COX A., BOSTOCK M.: Dissecting a Trailer: The Parts of the Film That Make the Cut. http://www.nytimes.com/interactive/2013/02/19/movies/awardsseason/oscar-trailers.html, 2013. 6

[Fek04] FEKETE J.-D.: The InfoVis toolkit. In *Proc. IEEE Information Visualization* (2004), pp. 167–174. 2

[Fla13] Flare. http://flare.prefuse.org/, September 2013. 2, 9

[GP96] GREEN T. R. G., PETRE M.: Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages and Computing 7* (1996), 131–174. 2, 6

[GW00] GUIMBRETIÉRE F., WINOGRAD T.: Flowmenu: combining command, text, and data entry. In *Proceedings of the 13th annual ACM symposium on User interface software and technology* (2000), ACM, pp. 213–216. 5

[HB10] HEER J., BOSTOCK M.: Declarative language design for interactive visualization. *IEEE Trans. Visualization & Comp. Graphics 16*, 6 (2010), 1149–1156. 2, 3, 4

[HCL05] HEER J., CARD S. K., LANDAY J. A.: Prefuse: a toolkit for interactive information visualization. In *Proc. ACM CHI* (2005), ACM, pp. 421–430. 2, 9

[HHN85] HUTCHINS E. L., HOLLAN J. D., NORMAN D. A.: Direct manipulation interfaces. *Human-Computer Interaction 1*, 4 (1985), 311–338. 2, 3, 5

[HR07] HEER J., ROBERTSON G. G.: Animated transitions in statistical data graphics. *Visualization and Computer Graphics, IEEE Transactions on 13*, 6 (2007), 1240–1247. 9

[Kos06] KOSARA R.: The US ZIPScribble Map. http://eagereyes.org/zipscribble-maps/united-states/, December 2006. 7

[KPHH11] KANDEL S., PAEPCKE A., HELLERSTEIN J., HEER J.: Wrangler: Interactive visual specification of data transformation scripts. In *Proc. ACM CHI* (2011). 9

[LAC*92] LUCAS B., ABRAM G. D., COLLINS N. S., EPSTEIN D. A., GRESH D. L., MCAULIFFE K. P.: An architecture for a scientific visualization system. In *Proc. IEEE Visualization* (1992), pp. 107–114. 2

[Per13] PERISCOPIC: Thoughts on Visualizing U.S. Gun Murders. http://www.periscopic.com/#/news/2013/02/thoughts-on-visualizing-u-s-gun-murders/, February 2013. 1, 2

[Pro] Processing. http://processing.org. 2

[RKMG94] ROTH S. F., KOLOJEJCHICK J., MATTIS J., GOLDSTEIN J.: Interactive graphic design using automatic presentation knowledge. In *Proc. ACM CHI* (1994), ACM, pp. 112–117. 3

[STH02] STOLTE C., TANG D., HANRAHAN P.: Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Visualization & Comp. Graphics 8*, 1 (2002), 52–65. 2, 3

[Tuf] TUFTE E.: Cleaning up Excel's poshlust graphics. http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=0002kk. 2

[Veg13] Vega: A Visualization Grammar. http://trifacta.github.io/vega, September 2013. 2, 4

[Vic13] VICTOR B.: Drawing Dynamic Visualizations. http://vimeo.com/66085662, February 2013. 3, 4

[VWvH*07] VIÉGAS F. B., WATTENBERG M., VAN HAM F., KRISS J., MCKEON M.: Many Eyes: a site for visualization at internet scale. *IEEE Trans. Visualization & Comp. Graphics 13*, 6 (2007), 1121–1128. 2

[Wea04] WEAVER C.: Building highly-coordinated visualizations in Improvise. In *Proc. IEEE Information Visualization* (2004), pp. 159–166. 2

[Wic09] WICKHAM H.: *ggplot2: Elegant Graphics for Data Analysis*. Springer, 2009. 2

[Wic10] WICKHAM H.: A layered grammar of graphics. *Journal of Computational and Graphical Statistics 19*, 1 (2010), 3–28. 2

[Wil05] WILKINSON L.: *The Grammar of Graphics*. Springer, 2005. 2, 4

© 2014 The Author(s)
Computer Graphics Forum © 2014 The Eurographics Association and John Wiley & Sons Ltd.