Design of a Visualization Design Environment (VDE)

William McGrath, Jeffrey Ericson
Computer Science Department
Stanford University
{wmcgrath, jericson}@stanford.edu

INTRODUCTION

There are many tools for specifying visualizations available today, ranging from easy-to-use programs such as Microsoft Excel [6] and Tableau [9] to powerful graphical toolkits such as D3 [2]. Excel and Tableau make it easy for a user to rapidly generate data visualizations and iteratively explore a dataset, but can be somewhat difficult to use and restrictive with respect to creating a layout and specifying the visual properties of a final visualization. Toolkits such as D3 are flexible and incredibly customizable, but a user must know how to program to leverage their power. This project explores the middle ground between ease of use and flexibility of expression in attempting to create a graphical tool that allows a user to precisely create visualization fluidly and easily. The project makes use of direct manipulation techniques as well as Flowmenus [5] to allow the user to specify the mapping of data to visual properties on the visualization itself. It also provides a property editor to allow the user precise control of the layout and style of the visualization.

This project re-examines how people create data graphics and attempts to make it a more fluid and intuitive process. Creating a data graphic is a process that involves a dizzying number of decisions. The mapping of data to visual properties of marks, color schemes, annotations, layout, and other factors need to be considered to create a successful data visualization. Supporting thoughtful consideration and precise, intuitive control of these factors is a challenging problem. At the same time, a tool must have enough defaults built in to avoid intimidating or frustrating new users.

This tool aims to be a graphical construction environment for D3 data visualizations. The target audience is anyone who desires greater control over data visualization they create and wants to share their work, but doesn't know how to code. Journalists and bloggers benefit from the fact that the tool runs in the browser and the D3 visualizations it creates are easily integrated into webpages. The tool is also helpful to anyone on the internet who engages in discussions around real life datasets such as the users of IBM’s ManyEyes [10]. These discussions are often most compelling when users “answer data with data”, i.e. reply to statements made about data graphics with additional data or different graphics of their own.

RELATED WORK

This project attempts to carve out a niche between visualization software that makes it easy to create a visualization, but difficult to tweak it, such as Tableau and Excel, and software that makes it easy to tweak a visualization, but difficult to create it due to requiring programming knowledge, such as D3 or Google Chart Tools [4]. We will attempt to learn from these systems in order to create a tool that permits both a high degree of flexibility in specifying a visualization as well as easy creation of visualizations without coding.

Excel does a good job of making it easy to create a graph from tabular data by simply selecting data and choosing a chart type. The same can be said for Many Eyes, although it is collaborative and web-based. The main drawback of this approach is that not all data is best displayed as one of their preselected common chart types. Tableau improves on Excel and greatly increases the expressive power of preselected charts by allowing more options that dynamically update as data associations change. Tableau also defines an algebra that is hidden from the user that expresses the relations between data columns. However, Tableau still requires the user to understand how the different tools, boxes, and slides surrounding the visualization area affect the visualization, which can lead to unintended behavior. Additionally, Tableau doesn't allow for more subtle and fine-grained tweaks of the visual properties of marks, such as stroke width or color, among others.

Visualization toolkits such as D3 and Google Chart Tools as well as lower level graphics libraries such as Processing [8] or OpenGL [7] allow incredibly fine-grained control over the visual properties of a visualization, but each have a drawback. Existing visualization toolkits require the user to be proficient with a programming language in order to use them. Graphics libraries require coding knowledge and they also lack the visualization framework and helper functions included in packages such as D3, which significantly increases the time it takes to create a complex visualization.

Last is TouchWave [1], a new system for interactive visualization of stacked graphs on mobile devices. It allows a user to manipulate and explore data in a stacked graph using touch gestures. Although the goal of the system is somewhat different as it focuses on content exploration as
opposed to graph creation, the minimalist user interface design informs similar decisions made on this project. In particular, TouchWave makes it easy to scale, combine, and separate different series in a stacked graph to allow for rough comparison, while providing rulers that the user can call up to more precisely measure values at points. However, since our system is geared towards creation of new data graphics, it allows for a greater variety of marks and more control of the layout and styling of those marks.

METHODS
This tool is designed to offer an intuitive graphical user interface to new users, while allowing the precision that more experienced users require. We employ a drag and drop interface for the creation of new marks and for the association of their visual properties with data columns. Marks are directly manipulated as much as possible, with precise visual specifications made possible via the property editor.

Visual Interaction Elements
The system combines several basic visual interaction elements to allow a user to create a data graphic.

Mark
A set of graphical elements that correspond to data elements. Their visual properties such as fill color, position, opacity, and size can be mapped to data values. Marks include Rectangles, Arcs, Scatter Points, Axes, and Text.

Column
A named set of values of the same type, one per data element. In practice, the column name is the first row of an imported comma-separated value (CSV) file and its values are the remaining rows of the column. Values are deemed either quantitative (if they are numbers) or ordinal (if they are text) by our system.
bound for each mark. When an option in the menu is hovered over, a second level menu displaying additional mapping options is shown. In the case of a quantitative mapping, it allows the user to choose a linear or logarithmic scale. In the case of a nominal color mapping, the user can choose from one of multiple pallets. This allows the user greater expressiveness in mapping data, while making the mapping options readily apparent. Dropping a column onto a property without choosing a sub-option simply assigns the column to the property using a default choice of scale.

The flow menus remember the current state of the mark by displaying properties that are already mapped in bolded text. This behavior is additive except for two special cases. A rect mark can have multiple data-mapped properties, but cannot have both its width and height mapped to data, by definition of a bar graph. If the user drags a column onto the width element in the flow menu, a previous data mapping to height will be dropped (and visa-versa). Likewise, an arc mark cannot have both its inner radius and outer radius mapped to data. If the user drags a column onto the inner radius in the flow menu, a previous data mapping to outer radius will be dropped (and visa-versa).

One important note about mark types is that they are not pigeonholed into chart types. A scatter point mark does not need to become a traditional two-dimensional scatterplot. If the user maps data to only one of its x and y visual properties, the result is a one-dimensional frequency chart. Likewise, an arc mark does not need to become a traditional pie chart. If the user increases the length of its inner radius in the property editor, the arc mark becomes a donut chart. This makes visual exploration easier, particularly compared to visualization systems like Microsoft Excel that ask for a specific chart type upfront.

Data columns can also be dragged onto special zones attached to each mark to label each data element with the data from that column. This allows the convenient creation of annotations as well as displaying an additional category or dimension on top of a visual display for a given column. Although a text label is created for every data element, the system uses a simple overlap detection algorithm to reduce clutter by hiding labels so that none overlap. If the user later scales the mark, the detection algorithm will update the number of labels visible (Figure 3). Thus, when marks are small, fewer labels will display to avoid cluttering, but when the user scales a mark to be larger, more labels will display (and visa-versa). This overlap detection update occurs as the user scales for the rectangle mark and once the user is finished scaling for the scatter point mark. It was visually disorienting to update the scatter point mark’s data labels during scaling.

The user can style the data labels by clicking once on any label in the group (which turns all labels in the group orange temporarily) and then using the property editor to format the color, font size, font stylings, or font type. Again, the simple overlap detection algorithm will run when the visual properties of data labels are updated. For example, if the user reduces the font size for a group of data labels, more labels from the group can be visible because fewer labels overlap. Additionally, the system provides fine-grained control for editing data labels. The user can style an individual data label by double clicking on it (which turns the label red temporarily) and then using the property editor. The user can also drag an individual data label to a new spot to override the default positioning assignment by the system. This fine-grained control of the color and position of a single label could be used for aesthetic reasons or to highlight a particular data element.

Axes are created in a similar manner as labels. Axis marks can be dragged on to drop zones on the side of a mark group. This creates an axis that is bound to that side of that mark and dynamically updates as the mark is scaled or the data is changed. The system also changes the number of ticks shown on an axis dynamically as the mark group is scaled to allow for greater precision in reading large charts and reduce clutter on small charts. Additionally, the axis marks are automatically labeled with the column name of the column that is mapped to that dimension. An axis has a default position, but the user can customize its position by dragging it further from the chart. For example, for an x-axis, the user can only drag it vertically, as to not distort its accuracy with respect to the data points of the mark.

Layout and styling of the elements is handled through a combination of direct manipulation of the marks and the use of a property editor. A user can move and scale any of the marks into appropriate positions. For more precise control of layout and styling, a property editor element is provided. The property editor shows all available editable properties for a given mark, while also indicating which editable properties have been mapped to data to prevent unintended overriding of those mappings. However, the user can still choose to override a data-mapped property in the property editor by assigning the property a constant value. Each mark has a blue “i” icon in its upper-right corner to the left of its delete icon. Hovering over this blue “i” icon will display a tooltip that shows all combination of mapped data columns and their visual properties.

As is the case with the menus, all of the available properties for a selected mark are immediately shown and available for the user to edit, removing the confusion of digging through multiple levels of menus for a given property. The property editor starts blank, but is populated when a user clicks on a mark. The content of the editor is context specific in that it only shows fields that are assignable for a selected mark. For instance, a rect mark does not have an assignable “inner radius” parameter, while an arc mark does.

Finally, text marks are available to be used for titles or additional annotations. Their text is directly editable and multi-line boxes are supported if the user presses the “Enter” key on their keyboard (by default, there is no
One important strength of the system is to allow the user to easily manipulate the positioning and display for a set of marks. To that end, the user can make as many marks as they wish on the canvas. The overall layout of the user interface was motivated by the desire to let users explore their data by making multiple marks, but always having the tools they need on the left side of the screen. Thus, the left side of the screen is fixed positioned and the right side of the screen offers ample space for exploration in the canvas (2000 pixels wide and 3000 pixels high), but not too much space as to get lost. The property editor also has a toggle button for guidelines to help the user position elements via direct manipulation.

As was previously mentioned, the user can also forgo directly dragging elements and specify the marks’ x and y positioning with exact values in the property editor.

RESULTS
We tested our visual development environment with a range of datasets: one of information about movies (including ratings from IMDB and Rotten Tomatoes, worldwide and US gross, and nominal information like movie title, director, and major genre), one on San Francisco Bay Area
weather for the last year (including rainfall totals and temperature by month), one on the 2012 Summer Olympics (including medal tallies and country’s population and GDP), and one on the 2012 NBA champion Miami Heat roster statistics (including statistics for the regular season and for the playoffs). Screenshots are in Figures 4 through 7.

Figure 4. Visualization using movie dataset. Data mapped includes production budget (x-axis), worldwide gross (y-axis), and genre (fill color). Text further customized with different fonts and sizes. Legend automatically generated.

Figure 5. Visualization using weather dataset. Data mapped includes high temperature (length) and number of rainy days (opacity). Data labels shown to the left of the y-axis (month name) and to the right of each bar (high temperature value). Text color customized.

Figure 6. Visualization using 2012 Summer Olympics dataset. Data mapped includes 2010 population (x-axis) and gold medals won (y-axis). Fill color and stroke color of dots customized. Annotations use customized text.

Figure 7. Visualization using Miami Heat dataset. Two arc marks include player name (fill color) and either regular season points score or playoff points scored (angle). Legend automatically generates with 2 columns.
Throughout the development of the system, feedback was sought from Professor Jeff Heer, Arvind Satyanarayan, and the course TAs. Their input was instrumental in shaping the system and interactions. A small user observation study was conducted by observing a member of the Stanford HCI group using the system when it was nearing completion. Additionally, many of the guests at the course poster session provided interesting opinions and concepts to explore in future work.

The user observation sought to evaluate the ease of use of the interface by allowing the subject to use the system with no formal explanation or training. However, the subject was a college-aged HCI researcher and was informed that the broad purpose of the system was to create data visualizations. What follows is a brief summary of the observation session.

**User Observation Notes**

- Subject immediately attempts to drag data columns onto the work area and notices that there is no effect.
- Subject next drags an arc mark icon, drops it onto the work area, and notices that it creates a basic arc group.
- Subject realizes the purpose of the mark icons and proceeds to drag a scatter mark onto the work area.
- Subject successfully assigns a data column to one of the two positional encodings of the scatter mark. As the data elements animate into position, the subject voices their confusion that all points began to move stating, “What did I just do?”

**Suggestions for Improvement**

A discussion of the other features of the system and suggestions for improvements followed. Although the sample size was small, the main takeaways from the study were that the drag and drop interaction was reasonably intuitive and that it was essential to expose the state and actions of the system to avoid user confusion.

We sought to address the “What did I just do?” problem with three changes. First, we added a hover-able information icon attached to any selected mark that would display an annotation with all of the “data - visual property” mappings currently associated with that mark. Next, we modified the axes so that they would be automatically labeled with the appropriate column name when they were attached to a mark group. Last, we changed the default state of the scatter mark group. Originally, scatter marks were by default set to have both their x and y positions associated with their implicit data index in the dataset. Thus, they would appear as a line segment with slope of negative one on creation (Figure 8). When a data column was assigned to one of the position encodings, the points would all move and create an unconnected line chart, confusing the user. The scatter mark was changed so that both x and y position were set to zero upon creation. Thus, when a position encoding is assigned, the points spread out to create a 1D frequency distribution and it is obvious that only one dimension has been assigned.

**DISCUSSION**

As we demoed and explained the system to the guests of the poster session, we had several takeaways. Encouragingly, many people seemed to resonate with the broad goal of our project, to make it easier to create stylized D3 data graphics and were curious about how the project would be made available, whether deployed on a website or as an open-source project. In allowing users to demo the system, we realized the value of a system identifying and displaying the type of data that a column represents. Since the system cannot yet handle a nominal value mapped to a quantitative encoding, it prevents it. However, since it does not announce or present this fact to the user or give an indication of the types of the various columns, its behavior in this situation can appear strange or anomalous. Also, when
users attempted to assign columns or axes to dropzones (Figure 9) we recognized the importance of Fitts’ law [3] in our drag interactions. Although we had experience with the system and could hit the somewhat small zones with ease, users struggled and were frustrated by the lack of feedback as to whether their positioning was correct. This effect was exacerbated by the large monitor for the demo during the poster session. We increased mouse speed to keep dragging fluid with the large monitor, but this speed increase coupled with the smaller relative size of the dropzones made dropping items with precision difficult.

We also had higher level discussions beyond the demo with the poster session attendees. One was about the spectrum of user freedom, from the ability to heavily customize to providing smart defaults to preventing users from making poor design decisions. The takeaway was that smart defaults should be used as much as possible, but in order to offer the end user as much customization as possible, only egregiously bad design decisions should be prevented. Choice of font color, for example, should be completely at the discretion of the user, regardless of coordination. However, misaligning axes or making the dot size of a scatter point too small should not be allowed.

Small multiples came up, which also led to topics of including the ability to bin, slice, and query the underlying data. Those items were all out of scope for our system in the given timeframe, but would be essential to expand the expressiveness of the system.

Additionally, questions arose on importing data and exporting the state of the design or exporting to an image format. Particularly since the target audience of the system is users without coding experience, the data import feature would need to be very user-friendly and flexible to handle missing data or poorly formatted data. Many parties also had different desires for exporting, from exporting D3 and JavaScript code to being able to export vector graphics. It was clear from discussions that our system would still be desirable to people who know how to code because of the potential time savings. Being able to generate D3 and JavaScript to allow further custom modifications or to expand on a template would be excellent features for power users. For users without coding experience, the primary functionality desired in an export feature seemed to simply be a way to embed their visualization online or export to an image format.

**FUTURE WORK**
The goal of this system is to test and evaluate the potential of different interaction techniques that can be used to create data graphics. Once the basic interaction techniques are settled, there are several directions that it would be interesting to pursue in making this project into a tool that could be deployed for public use. The least difficult of these directions is the addition of features such as an undo history and a way to export finished graphics. Additionally, while basic properties of the marks are supported, more mappings for existing marks could allow for greater expressiveness.

Of particular interest are possibilities for supporting mark to mark interactions. One can envision creating stacked bars by combining two bar marks or binding properties of one mark to a property of another mark to allow for combinatory expressiveness in a similar way as Tableau does. In this way, the tool could offer preset layouts such as bar or line graphs for convenience, but the marks that make up these graphs would in fact be editable compositions of simpler marks. The composition marks could be individually edited and recombined to allow the user to break out of the taxonomy of preselected chart types.

Last, D3 makes adding animation and interaction to graph elements easy for its users. There is a need for graphical tools to allow users to specify interactions and animations without coding. It is a challenge, as there are countless ways to interact with data graphics, but such a system could benefit from a library of common, pre-made interaction techniques. Such a tool would allow many more people to create compelling interactive graphics such as those featured in The New York Times.

**REFERENCES**
3. Fitts, P. M. 1954. The information capacity of the human motor system in controlling the amplitude of movement. Journal of experimental psychology, 47(6), 381.
7. OpenGL. http://www.opengl.org/