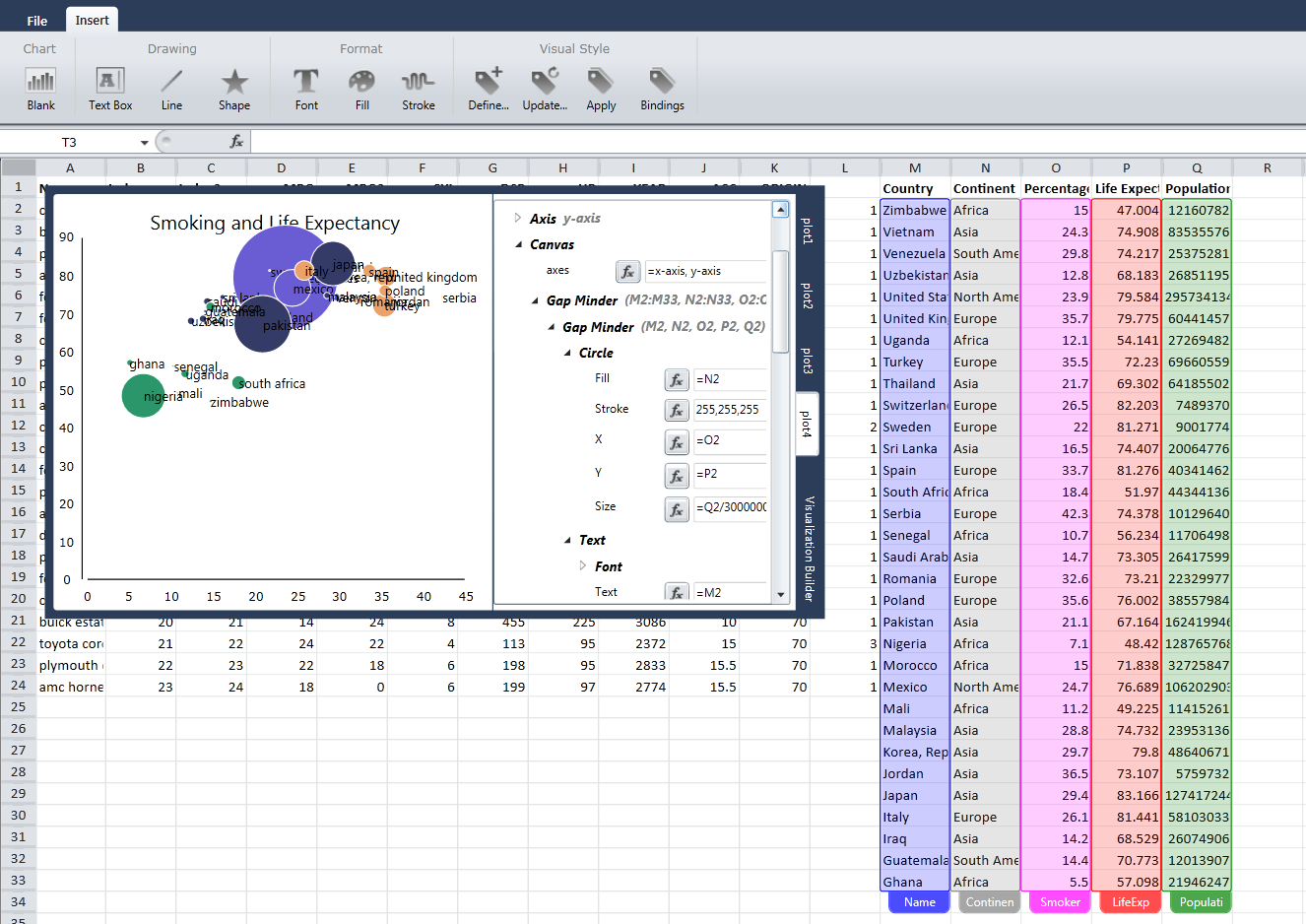
Vis-à-vis: A Visual Language for Spreadsheet Visualizations

Danyel Fisher, Steven M. Drucker, Roland Fernandez, Xiaoji Chen

Abstract

Finding ways for information workers to easily create and modify visualizations that display their own data has been a long time goal within the visualization community. We describe Vis-à-vis, a declarative language for defining and extending visualizations directly within spreadsheets. Vis-a-vis allows users to directly bind data and formula to the visual attributes of an extensible set of visualization primitives. The visualizations that Vis-a-vis creates can be shared and modified easily, allowing users to modify existing visualizations. This approach allows users to select visualizations from a gallery, to customize them easily, or to create novel visualizations. The approach leverages familiar formulas and data from spreadsheets. We prototype a system that uses this language, and use it to build a number of standard and custom visualizations, and gather formative feedback from a small user study.

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



**Figure 1:** *Vis-à-vis in action. The* Bubble Chart *style uses four columns of data to specify color, size, and location for circles, which are all bound to variables and formulas. (Chart enlarged for clarity.)*

# Introduction

Currently, most spreadsheet systems including Microsoft Excel, Google Spreadsheets, and Open Office have limited extensibility mechanisms for visualizations. If a user desires a visualization that is not built in to the system, the user usually needs to resort to several non-optimal solutions. They can export the data and import it into another system such as R and *ggplot* [Wic09] that supports their desired visualization, though that can require additional expertise and working styles separate from those with which they are familiar. Also, once the data has left the spreadsheet, the user may not be as comfortable exploring the data.

Alternatively, they can export the data and write a visualization themselves, perhaps using a library such as Prefuse [HCL05], the Infovis Toolkit [Fek04] or Protovis [BH09], though many knowledge workers may not have the coding expertise or the willingness to build new visualizations even given such assistance.

If they are fortunate, they can find an add-in which supports the visualization they desire, though that can have security and privacy implications.

Vis-à-vis is a system that allows conventional and novel visualizations to be built within the spreadsheet using common spreadsheet metaphors. By leveraging spreadsheet concepts and basic direct manipulation capabilities, we can greatly increase the kinds of visualizations that can be incorporated into spreadsheets without resorting to new programming APIs or toolkits. Spreadsheets are a familiar context for computation and end-user programming [ABE09].

The approach presented here is inspired by the declarative style in such systems as the Grammar of Graphics, Protovis, and ggplot. Vis-à-vis tightly couples the cells of a spreadsheet to the attributes of graphical marks in the visualization using the same kind of formulas already familiar to spreadsheet users. Changes in cells and formulae are reflected instantly, allowing an incremental working style for creating and modifying visualizations. The coupling can be saved as a ‘visual style’ and subsequently be reapplied or modified so that end-users can base visualizations on an existing visual style or combine multiple styles together. In fact, most of the time, they might use these pre-authored styles for constructing their visualizations, but the formulas that map data to visual attributes in these styles are discoverable and modifiable so that a user can specialize the appearance of the visualization for their own needs.

The primary contribution of this work is an end-user programming environment that maps the spreadsheet paradigm of cells and formulae to visualizations by the use of visual styles. When a particular visualization is not available, users do not need to export their data from the spreadsheet into another system: they can find it from a gallery of shared visualizations; modify it from an existing visualization; or build it within the spreadsheet environment themselves.

# Related Literature

Vis-à-vis blends concepts from drawing systems and spreadsheet tools. In this section, we discuss several of the techniques and technologies that have inspired its design. We begin by discussing a brief history of linking visualizations within spreadsheets. We then discuss two different strategies for visualization: one uses data binding to tie data to underlying object states; the other uses direct manipulation.

## Incorporating visualizations in spreadsheets

Spreadsheets and visualizations have a long history: spreadsheets have been used to store the data that the visualization will use, of course, and so make a natural place to host visualizations. (Conversely, many visualization packages provide a datasheet view of the data.)

Several systems have provided black-box visualization tools. WebCharts [FDF\*10] allows users to dynamically add visualizations to spreadsheets; however, those visualizations needed to be created in a separate program. Similarly, IntelligentPad and IntelligentBox [IT09] allow users to inject data from Excel into web-based 3D visualization services. None of these systems had a built-in way to edit visualizations and all of these systems treat the visualizations essentially as ‘black boxes’ which are passed data from the spreadsheet and use pre-authored code to create a visualization. Vis-à-vis chooses to expose the building blocks so that people can re-use or significantly alter an existing visualization, or build a new one from scratch.

A different tradition of research has created spreadsheets *of* visualizations. Piersol’s ASP [Pie86], Levoy’s *SI* [Lev90] and Chi *et al’s* SIV [CRB\*98] incorporate spreadsheets with visualization, albeit in a very different perspective. SIV allows users to create a matrix of visualizations by using formulas to invoke the spreadsheet. For each cell, a user can select both the visualization that will fill the cell, and the formula that defines the data that goes within that cell. This makes it very easy to compare small-multiple views with different data or different parameters, or to create multiple views of the same data. Vis-à-vis, in contrast, is designed to simplify the creation of a single visualization, although it can be used to create small multiples. In TableLens [RC94], the spreadsheet itself *is* the visualization: cells that are out of focus are summarized as bars or dots, so that groups of cells can be looked at as small charts.

## Direct manipulation and visualizations

In order to get visualizations that look exactly as the end-user desires, many systems allow rapid updating based on data and the ability to select parts of the visualization and change attributes such as line color, width, and style. To date, direct manipulation has primarily been limited to changing these visual attributes, though some systems have been built that directly supported users in drawing data visualizations. One such system, GraphSketcher [SS09] allows users to sketch graphs and diagrams; GraphSketcher tracks where users have placed edges, and maintains these as constraints: for example, if a user draws two crossed lines over a given point on the axis, those lines will remain crossed at that point even if the axis is shifted or moved. GraphSketcher has only a limited notion of data, and cannot be linked to data sources, and is primarily focused on line charts.

Commercial tools like Microsoft Visio and Adobe Illustrator both allow users to create drawings, including data visualizations. In Illustrator, though, the process is limited; only a few visualizations are supported, and changes to the data do not automatically update the visualization. Visio can allow shapes to be bound to data and formulas, but the interface is not optimized for general visualizations.

## Data-Bound Drawings

Vis-à-vis is based on the idea of using data to generate visual objects. In this, it draws from the SAGE family of tools from the mid-1990s, including SageBrush and SageBook [RKM\*94]. While the original SAGE, like APT [Mac86] attempts to create visualizations automatically based on a dataset, SageBrush allows users to guide its process. Users can drag drawing objects—text boxes, shapes, and lines—onto the canvas. If the user does not specify bindings, the system will automatically select and generate a coordinate system, using SAGE, but taking the drawing objects as constraints.

The user can bind attributes of these objects to data: for example, they might specify that the y-coordinate of a shape be bound to a particular column of data. When SAGE attempts to lay out the diagram, it will start by both using these drawn objects, and with the y-coordinate assigned to the data value. SageBook allows users to share their created visualizations; collaborators can then apply their own data to the visualization.

Vis-à-vis shares many of these concepts. Like SageBrush, Vis-à-vis allows users to create visualizations through a template model. Vis-à-vis does not attempt to automatically assign metadata fields, preferring to give users full control over how their objects are placed on the canvas. In SageBrush, users place objects onto a template canvas which controls the visualization; in contrast, Vis-à-vis, has users place objects directly onto the same canvas on which they will be visualized: the canvas *is* the display surface. Lastly, many data fields require transformation before they are ready to be visualized; Vis-à-vis includes equations in the form of spreadsheet formulas very naturally, without creating auxiliary data fields.

Like SageBrush, Improvise [Wea04] allows mapping of data to visual attributes using interpreted or compiled code though not in a spreadsheet environment or using spreadsheet metaphors.

NoPumpG [Lew90] allows users to create visualizations within spreadsheets. Users place controls, and then bind the visualization to cells, which contain names and formulae. For example, the cell “line.y” (meaning the y attribute of the line) might be bound to the formula “y0+y”, meaning the values of the cells “y0” and “y”. While this system uses cells and formulas, it loses its ability to act as a spreadsheet: there is no structure enabling users to work on parallel operations or multiple items of data at once, nor to use relative addressing. Vis-à-vis uses a true spreadsheet to enable this sort of structured behavior.

Vis-à-vis allows for direct manipulation of items, which can then be encapsulated in visual styles that can be reused within a single and across multiple visualizations.

# The Design of Vis-à-vis

Vis-à-vis is meant to find one possible balance point that trades off the relative advantages of writing raw code, using a drawing program and using the techniques already familiar to most spreadsheet users – formulas based on cells within the spreadsheet.

## Tradeoffs of Visualization Systems

In the literature section, we began to identify important issues in designing a visualization system. Here, we examine these attributes in more detail. We compare them, in general, between systems based on writing code, drawing programs, and packaged visualizations, such as spreadsheets or desktop applications. Our design is intended to allow Vis-à-vis to balance these attributes:

*Flexible:* A flexible toolkit gives a user choices in developing a visualization. While different tools provide different degrees of customizability, here we mainly distinguish between tools that have a fixed set of visualizations, and those that allow users to create or share a broader set.

*Does not require new languages:* Writing code requires fairly specialized skills that many otherwise-capable data analysts may not possess. While some packages allow a variety of approaches (‘R’ has plotting functions ranging from a grammar-of-graphics to command-line ‘plot’), we generally mean the question of whether creating a new visualization requires development experience. While some consider writing formulas within a spreadsheet an alternate form of coding, it is usually seen instead as a familiar form of end-user programming [ABE09]. Spreadsheet users have typically been exposed to formulas.

*Bound to data:* While most visualization tools make it very easy to visualize data, drawing tools are usually *not* linked explicitly to a data source.

*Reusable:* Part of the ability to bind data implies an ability to assign new data to an existing visualization. In general, we are interested in whether there is a way for a user to identify a past visualization of interest and assign new data to it. Pure drawing packages never create visualizations that are reusable in this way: the image might be saved, but the illustration needs to be otherwise re-generated for new data.

*Modifiable:* In addition to reassigning the data in a visualization, users may wish to customize a visualization in both subtle ways (such as tweaking colors) and more significant ways (altering the layout based on a new algorithm). Protovis [BH09] is specifically designed to be accessible, which helps users understand how a visualization was created, and how to create the next version.

*Sharable:* Users should be able to share the components of their visualization with each other, including allowing another user to reuse the visualization with different data, or to extend an existing visualization. Sharability is largely implied by modifiability, but requires the ability to distribute the visualization’s algorithm separately from its data.

*Directly Manipulable:* Users ofdrawing tools who wish to move an object or change its color are accustomed to merely dragging it to its new place or dropping on a new color palette. In contrast, developers who wish to change the color of an object are more likely to need to recompile their code. Excel and other visualization packages permit some direct manipulation, but usually not as powerfully as dedicated drawing programs.

Table 1 highlights the ways these tradeoffs manifest in different categories of tools. We also include the target audience for each category, separating out the skills demanded by each type of tool.

**Table 1**: Tradeoffs of writing code, drawing programs, and visualization packages. 🗸 supports an attribute, 🗴 lacks it, and ? means “in some cases” or “limited.”

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Code** | **Drawing Programs** | **Visualization Packages** |
| **Audience** | Developers | Artists | Info Worker |
| **Flexible** | 🗸 | 🗸 | 🗴 |
| **No code** | 🗴 | 🗸 | 🗸 |
| **Data bound** | 🗸 | 🗴 | 🗸 |
| **Reusable** | ? | 🗴 | 🗸 |
| **Modifiable** | ? | 🗴 | ? |
| **Sharable** | ? | 🗴 | 🗸 |
| **Direct  Manipulation** | 🗴 | 🗸 | ? |

## Design Goals

Vis-à-vis attempts to find comfortable compromises between all of these points. Vis-à-vis is designed to be *flexible*, allowing users to create visualizations that have not been seen before, or to add their own twists to existing visualizations; these visualizations should be able to be created by people using the same metaphors that they are used to from working with their data and not develop them from scratch in another system or language. Vis-à-vis is intended to be *data bound,* andfor it to be possible to *reuse* components with different data. These components can be *shared* with other users. Lastly, Vis-à-vis contains elements of a *direct-manipulation* environment; users can directly modify the visualization through selection and commonly used metaphors such as menus and palettes.

Critically, Vis-à-vis is designed to be incorporated into the spreadsheet, allowing users to interact with the visualizations in familiar ways, and in order to ensure that users can seamlessly move from editing and manipulating data to visualizing it. This builds a complementary workflow: a user may generate data using a formula and subsequently assign that data to an attribute of the visualization.

Users can also take advantage of the spreadsheet to compute intermediate values and transform the data, when needed, to better fit the visualization data binding. They can also work incrementally; their actions—such as entering formulas and selecting from menus—are immediately reflected in the visualization.

## The Vis-à-vis Language

Vis-à-vis is a declarative language incorporated into a spreadsheet. The language represents a visualization as a hierarchy consisting of **marks** placed on a canvas. The root of the hierarchy typically contains axes, titles, and a canvas. Marks may be primitive objects (such as text, shapes, and images), or compound objects made up of primitive objects. Each mark has a series of attributes: at the least, a position and size; they may have additional dimensions, as applicable. For example, text has a Font attribute, and almost all shapes have a Fill attribute; images have a ‘source’ attribute, and rounded rectangles might have a corner radius.

Every field on a mark has a value. The value can be a **constant**, a **variable**, or a **formula**. Each field starts with a constant, default value; while an object may have many possible fields, the majority are initially hidden from the property list. When a field’s value is changed through direct manipulation (such as with the mouse or palette), it then is shown on the property list.

As Vis-à-vis uses familiar spreadsheet language, variables and formulae are addressed with spreadsheet values. For example, a “height” variable marked as "=*C2"* means that the height of a shape takes the value of the spreadsheet in the third column, second row; the formula



Figure 3: A palette of some available styles, with named parameters

=SIN(C2) + D3

will return the sine of the value of that cell, plus the value of the cell immediately below and to the right of it.

Figure 2 illustrates both the visualization hierarchy, and the ways that formulae are linked to cells. The formulas themselves are not stored in spreadsheet cells, but rather in a panel that annotates the canvas. The overall structure of the system is shown on Figure 1.

### Formulae and Relative Referencing

As with most spreadsheets, references to cells are *relative by default*, not absolute, which means that when a formula is copied, it propagates its values appropriately. If a style that applies to a mark cell C2 contains the formula "*height* =C2", when the style is applied to cell C3, then formula will be updated to "*height* =C3". To prevent this from happening, the user can use ‘dollar’ notation; *height = C$2* will fix the height at the value of C2 for all cells in the C column that the visual style is applied to. Relative and absolute references can be combined; setting the formula on mark’s *height* to *C3/C$2* set the height to be the fraction of C2 that is set by the given cell.

Users can take advantage of relative referencing to allow references to ‘previous’ and ‘next’ values. For example, if a user wished to set a point to be red if its value is less than the one before it, they might enter the formula color=IF( C2 < C3, “red”, “green”)

### Visual Styles and Compound Shapes

A style is a mechanism for saving the mapping between spread-sheet elements and shape attributes. It is the basis for re-use across elements within a single visualization or across multiple visualizations. We will describe visual styles in detail in the walkthrough section below.

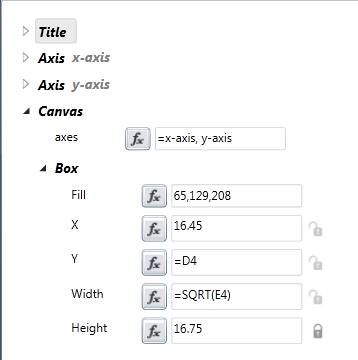


Figure 2: The visualization hierarchy. The visualization has a title, two axes, and a canvas; the canvas has a box, which has some fields assigned with variables and formulae.

Visual styles also allow for the creation and mapping of compound shapes with their own parameters. While Vis-à-vis comes with a palette of primitive shapes, a user is likely to want to replicate their primitives across the board with arguments. For example, error bars can be understood as four shapes: a top and bottom cap, a line in between, and a marker at the mean. Rather than addressing these shapes as four separate elements filled with formulae, cell references, and constants, a user can store all of these together as a style. The style might take only parameters “index”, “mean”, and “standard deviation”, allowing users to easily remember how to apply it.

When a shape with fields is added to a style, constants remain constant. Relative cell references, including those that appear within formulas, become named parameters to the style.

Styles encapsulate the information about how data should be rendered; a visualization can therefore be understood as one or more visual styles, bound to one or more sources of data.

### Sharing and Reusing Visualizations

The style mechanism allows visualizations to be re-used and modified by later users. Within the Vis-à-vis interface, users can create new styles either from scratch or by modifying existing styles; they may update existing styles, or bind an existing style to ranges of data within the spreadsheet. With these tools, users can share and publish visualizations. In figure 3, the user is selecting which visual style to apply to a visualization.

### Brushing and Linking

All data is linked between the spreadsheet and the visualization, so that when cells in the spreadsheet are selected, any marks that use these cells in the visualization are also selected. Likewise, selecting marks in the visualization will select cells that are used in the visual style for that element. In a simple bar chart, for example, the user might decide to select some set of data values in the spreadsheet. The corresponding bar would be selected; the user could then change the fill color with the Fill Menu. Conversely, in order to discover the data that created a mark, a user can simply select the mark to have the underlying data selected.

The system also supports tooltips. When the user hovers over a mark, the system displays all the parameters that are populated by formula, and their values.

### Changing Data or Formulae Updates the Visualization

Users of many programming languages, including visualization languages, expect to make changes to code, and then execute the resulting system. In contrast, users of spreadsheets expect to see formulae update across the sheet with changes. Vis-a-vis maintains a dependency graph of formulas to support automatic updates of dependent targets (cells and visualization attributes) when data is changed. Therefore, the visualization is changed when a formula, value, or attribute is altered.

As a result, Vis-à-vis can be used to create a visual ‘dashboard’, with the visualization changing and updating as values in the spreadsheet change.

## A Walkthrough of the Interface



Figure 4: Assigning a variable to fields. A bar in the chart has width 1, a Y value of 0, an X position of “=A2” (its index) and a height of “=B2” (its value). Later on, we show a function, *rowindex()* that lets us skip using a cell in the spreadsheet for this purpose.

We discuss the design of the system by walking through the process of creating a simple visualization. In this walkthrough, we create a column chart, and then modify it to be a ‘waterfall’ chart, a common business graphic that shows cumulative values.

While in this example we create a visualization from a blank canvas, users will more often modify and enhance an existing visualization. This walkthrough, however, allows us to better illustrate the capabilities of the system for creating a visualization that does not currently exist in most spreadsheet systems. The initial data has two columns, storing an index (column A), and corresponding values (column B). A1 and B1 contain headers, so the data runs from A2 through A21, and B2 through B21.

The accompanying video figure shows a similar process, and creates a similar pair of visualizations.

### Creating and editing an object

When the user begins Vis-à-vis, they are given a blank canvas with a default pair of axes. The user can use standard drawing tools to create and drag arbitrary lines and objects onto the canvas. The objects can be moved, resized, and colored using standard direct manipulation techniques.

Unlike previous spreadsheets-of-visualizations [CRB\*98, PIE86], Vis-à-vis instead creates its visualization in separate drawing area (Figure 1). The drawing area consists of both the drawing canvas and the visualization builder. The visualization builder window (Figure 2) contains a list of the objects that are currently on the canvas. Each object can be opened and attributes of the object can be modified via a property explorer. Attributes change as the user modifies the object on the canvas – for example, as the user resizes a rectangle, its updated width and height values are reflected in the property explorer. Attributes can also be set to a constant value or formula by typing the value into the property explorer.

### Screen space and coordinate space

A pair of *x* and *y* axes are included with a canvas. The default behavior of these axes is to automatically scale the objects in the visualization to lie between the minimum and maximum ranges of all the objects on the canvas. This behavior can be overridden to include a manually set minimum or maximum (or both) or the axes can be converted to logarithmic scales. In addition tick marks can be set to appear at automatic or manually set intervals. The axes are also marks; they default to automatic scaling, and fit all values on the chart. The user can override them by setting the appropriate fields.

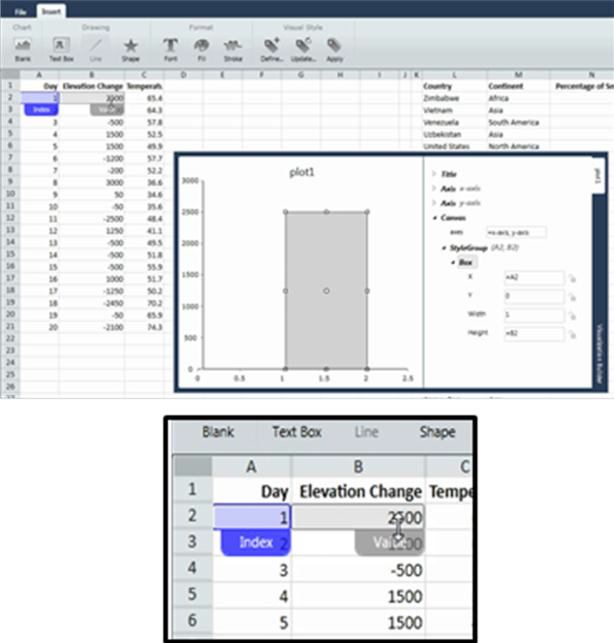


Figure 5: Styling a chart. The Column Chart style takes two parameters which are propagated to the drawn Box. The detail belows shows the columns bound to the parameters: Index in blue, Value in gray. The regions can be dragged to bind more values to the style.

For many visual styles, some attributes of the drawing need to be specified in absolute (screen) coordinates and others need to be scaled by an axis. For example, a Y axis on a bubble chart might be denominated in millions of dollars. The radius of a circle drawn on that chart will not be counted in millions of dollars, however, but rather in screen pixels; the typeface will be 20 pixels high (not 5 million). Vis-à-vis allows a user to label whether a value should be in screen or axis coordinate space with a ‘lock’ icon (see figure 2). Beyond these convenience measures, of course, a user can also create a custom conversion to transform coordinates.

### Binding data to the object

An attribute in the visualization builder window can be bound to a value in the spreadsheet following familiar spreadsheet metaphors: the attribute acts just like a cell, and so can be assigned cell values or formulae. When the value in a cell changes the attribute automatically updates. To create a column chart, we first create a box shape to be our prototypical bar. This bar will be used as a visual style when we bind to multiple rows of data. This bar will initially be bound to row 2 of the spreadsheet (figure 5 shows the data used in this example). We bind the *X* value of the bar to the *index* column, cell *A2,* and the height to the *value* column, *B2*. Figure 4 shows a single bar with two variable values.

### Defining a visual style

Once an object’s fields have been assigned to values, the fields can be preserved as a style. Having constructed a first column for a bar chart—fixed width, height relative to a column, x location based on index—we can then store a style, named *BarChartStyle,* taking parameters *index* and *value*.

In our example, we want to replicate this single bar for each row in the chart. A dialog allows us to define a new visual style, which assigns each variable to a parameter. In this example, both A2 and B2 are variables; we can assign them to parameter *Index* and *Value*. In Figure 5, we create a chart style.

### Completing the Column Chart

Once a visual style—in this case, the *Column Chart* *(index, value)*—is created, it can be extended down its columns to get a complete chart. Figure 6 extends the style from Figure 5 all the way down the column, applying it to all cells from A2:A21 and B2:B21. This causes a copy of our prototypical bar to be created for each row of data and has the effect of copying the bar across to create a bar chart. Alternatively, the user could have selected the desired columns and chosen from the Apply menu to apply the designated style. Note that as the visual style is applied to each element in the selection, the formulas that are used in the style are updated based on the relative referencing rules discussed in section 3.3.1.



Figure 6: Completed bar chart. The visual style has been applied to all cells in the block A2:A21, B2:B21.



**Figure 7:** The completed waterfall chart. The style has been updated and renamed to Waterfall.

### Waterfall Chart

So far, we have shown only how to reproduce a very common visualization, found in almost all packages. We demonstrate the flexibility of the Vis-à-vis toolkit by creating a less common visualization, the waterfall chart. A waterfall chart is a business visualization used to track cumulative profits and losses. (We demonstrate other less common techniques in figures 8 and 9).

Waterfall charts are a simple variation on column charts, but have two important differences. First, each bar grows or shrinks from the top of the previous bar to show a running total; second, each bar is colored based on its height: positive bars are green, negative are red. Waterfall charts can be difficult to produce in many visualization packages, as they require a cumulative sum; they thus highlight one particular virtue of using a spreadsheet metaphor.

When a user sets a bar to a specific color, a “Fill” field is added to the visualization builder. Vis-à-vis interprets both RGB triplets and strings as colors, and so users can assign a fill color to be the result of a formula. In the running example, “Fill” is set to the formula =IF(B2>0, “green”, “red”).

If each bar is to be a running total—that is, is to grow from the column before it—then the baseline of each bar must be at the cumulative total so far. We express this by selecting a bar, and assigning its Y (that is, base) value to a running sum. In this example, we are editing the bar that corresponds to row 9, and so we assign its Y value to =SUM(B$2:B8). B2 will be treated as an absolute value, while B8 will be treated as a relative one.

Last, we update the visual style to match, allowing the chart to update. The completed chart is a waterfall chart, shown in Figure 7. This uses the same data as Figures 5 and 6.

Many waterfall charts use special colors for the initial and final bars. If we wish to add that to the style, we would simply change the Fill’s formula to include a nested =IF statement; first to see if the shape is the first or last row of the selection and color them appropriately, and subsequently use the existing =IF clause.

## Extending Spreadsheet Formulae

It was our goal to leverage existing spreadsheet concepts as much as possible when designing Vis-à-vis. There are, however a number of small extensions that are helpful in making visual styles generalizable and sharable. Many of these involve more generalized ways of referencing the selection range to which a visual style has been applied. Table 2 shows the additional functions. In general, all of these selections are relative to the current bindings.

**Table 2**: Additional formulas added to the spreadsheet system

|  |  |
| --- | --- |
| rowIndex | 0-based relative index of the row. This a common helper function obviates the need for a column of values kept soley as an index (such as “day” in Figure 5) |
| rowCount | The number of rows in current context. Of particular use in radial layouts where we want to equally distribute the rows around a circle. |
| seriesIndex | The index of the series. Allows for application of a style to multiple columns of data without manually assigning a visual style to each column. |
| seriesCount | The total number of series. |
| current-Column | Returns the current full column of selection as a range |
| lastSeries-Value | Returns the final series value |
| lastSeries-Column | Returns the final series column |
| previous | Returns the previous cell relative to the current rowIndex. |
| Previous- Shape | Returns the formulas for the previous *shape*. |

## Using Vis-à-vis to Create a Chart

Vis-à-vis has been used to reconstruct a number of standard and non-standard charts. Figure 1 illustrates a *bubble chart,* in which each bubble takes on values for size, location, and color. Note that *size* is computed as a formula, and that the lock icon next to it is closed: this indicates that size is locked to screen coordinates. (A more complex bubble chart, with labels, can be seen in the accompanying video.)

Table 3 lists a number of common visualizations along with the visual styles for each element needed to create them.In that table, we show both the subset of the forum

Table 3: Several examples of vis-à-vis charts and the formulae that they use.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Marks Used & Code** | **Type** | **Marks Used & Code** |
| **Column** | **Box** X = 0.75 \* rowIndex() Y = 0  Width = 0.5  Height = A2  Fill = 220, 220, 220 | **Clustered Column** | **Box** X = 1 + RowIndex()  Width = 0.25  Height = B2  Fill = 128, 128, 128  This has two shapes in the style, the first exactly as above, and the second series sets the X locations to ‘1 + rowIndex()’ and the Height to a cell in the second column of the spreadsheet. |
| **Stacked Column** | **Box** X = 0.75 \* rowIndex() Y = **A2**  Width = 0.5  Height = B2  Fill = 128, 128, 128  The second series starts its boxes at A2, the height of the first series | **Scatter Series** | **Circle** X = A2  Y = B2  Size = 20 **{L}**  Fill = 0, 255, 255, 255  Stroke = 128, 128, 128  The size of the circles is locked to pixel space to ensure that they are not scaled to the axes. |
| **Line Series** | **LineSegment** X1 = if( rowindex()==0,  1,rowindex()) Y1 = if( rowindex()==0,  B2,previous(B2))  X2 = if(rowindex()==0,  1,1+ rowindex())  Y2 = B2  ‘Previous(cell)’ retrieves the value of the cell at the previous row. Each line is assigned to its own style | **Area Chart** | **Area**  X1 = if( rowindex()==0,  1,rowindex()) Height1 = if( rowindex()==0,  B2,previous(B2))  X2= if(rowindex()==0,  1,1+ rowindex())  Height2 = B2  The code is simiar to the line series, but the area mark has different parameters |
| **Pie** | **Wedge**  Angle =360 \*  A2/max(1,sum(A$2:A$6)BaseRotation = if( rowindex == 0,  0, prevShape(“Angle”) +  prevShape(“BaseRotation”))  BaseRadius =   if( rowindex() == 0, 30, 0)  Opacity =  (1.0+rowIndex())/  max(1,rowCount())  The helper function prevShape references the formulas from the previous rows. This is similar to previous, but references the formulas in the visual style instead of the cells in the spreadsheet. | **Donut** | **Wedge**  Angle =360 \*  A2/max(1,sum(A$2:A$6)BaseRotation = if( rowindex == 0,  0, prevShape(“Angle”) +  prevShape(“BaseRotation”))  BaseRadius = 0 InnerRadius = 40 OuterRadius = 40 Opacity =  (1.0+rowIndex())/  max(1,rowCount())  Like the pie chart above, but with an inner and outer radius defined for each ring. |
| **Donut** | The first series is the same as the donut chart above. The second series adds:  **Wedge**  Angle = 360\*B2/max(1,sum(B$2:B$6))  InnerRadius = 80 | **Picture Graphic** | **Image**  X = 50 + count(A$2:A2, A2)\*25 {L} Y = 280-(25\*(A2-1))  Width = 20  Height = 20  FileName = Images.Whitebody.png  Color = if (B2==’sick’, ‘green’, if(B2==’vacation’, red’, ‘blue’))  A non-standard chart that uses colored icons to represent time away from work (green for sick, red for vacation, and blue for business trip). |

The system is flexible enough to create a wide variety of visualization types. In Figures 8 and 9, we show three different visualizations that are not common today.

In Figure 8, we implement parallel coordinates. While it is straightforward to create a parallel coordinate diagram with a fixed number of coordinates, the core techniques of spreadsheet languages do not have the notion of relative placement needed to compute which column we are in. In Figure 8, we have implemented parallel coordinates without the functions in Table 2: that is, by simply hard-coding precisely three different sets of lines. In order to generalize to an arbitrary number of columns, we use the helper formulas seriesIndex(), seriesCount() and currentColumn() discussed above.

Figure 9a shows a “hi-lo/open-close” stock chart. This chart represent, in miniature, a stock’s trajectory through a time period: the ticks on the left and right indicate the opening and closing values, while the bar runs from the low value of the day to the high. The chart is constructed using a visual style that maps four columns of data onto three “Box” shapes. This is a good example showing how multiple primitive shapes can all be mapped within a single visual style to the input data.

Figure 9b shows a radial visualization that connects countries that have been laid out on a circle.

# Implementation Status

Vis-à-vis is currently implemented as a prototype. It implements a basic spreadsheet, with a robust formula vocabulary and a dependency graph of formulas. The current prototype is not currently intended to scale to massive size datasets; it is an output tool for information-worker tasks. As a result, performance has not been an issue; in our tests placing up to a few hundred objects with as many formulae and attributes, it runs at interactive speeds.

# Preliminary Study

To validate our design, we wanted to understand whether creating visualizaitons within a spreadsheet paradigm made sense to users who were already familiar with spreadsheets, but not with creating custom visualizations. We carried out a preliminary study to examine these questions.

The user study consisted of four stages in order of increasing complexity. Each of the first three stages introduced capabilities of the system to users while they worked on solving a problem, and built on the techniques taught in the previous stage. In the final stage, users built a visualization without guidance. The stages roughly followed section 3.4 above: participants built a visualization from scratch by adding a rectangle to the canvas, binding data to the rectangle, defining styles, and applying styles to multiple rows. They then modified the style from a bar chart to a waterfall chart. The users then built a bubble chart from scratch.



Figure 8: Parallel coordinates implemented in Vis-à-vis

The directions gave the users a sequence of tasks, but did not provide instruction or training. Participants were asked to think aloud, and intervention was kept to a minimum. The study explored both usability aspects of the tool, including which parts of the interface were discoverable or interpretable and which parts were not; and also to understand how users scaffolded existing spreadsheet skills to this system. In addition, the think-aloud protocol helped us understand how users were learning about the system.

## Results

We recruited five information workers with familiarity with Excel spreadsheets and who had previously used Excel charting. All participants had experience with the existing methods of charting within Excel; two participants use Excel to visualize data extensively. One of our participants did have programming experience, which included building custom visualizations in Excel from scratch using Visual Basic.

All the participants were able to grasp the idea of binding data to attributes of shapes on the canvas; one of the five participants needed prompting to realize that he could use cell references as fields. Several participants had difficulty in the individual stages for modifying the column chart into the waterfall chart (stages 2-3); however, all participants were able to generalize the principles and apply them to the creation of the bubble chart (stage 4), with no prompting. The time to complete stage 4 ranged from 3 to 14 minutes.

Interestingly, the user that required the most prompting early on was the user that was most experienced with programming, having already created visualizations with Visual Basic. Once he caught on to the idea of visual styles, however, he was the fastest at creating the bubble chart.

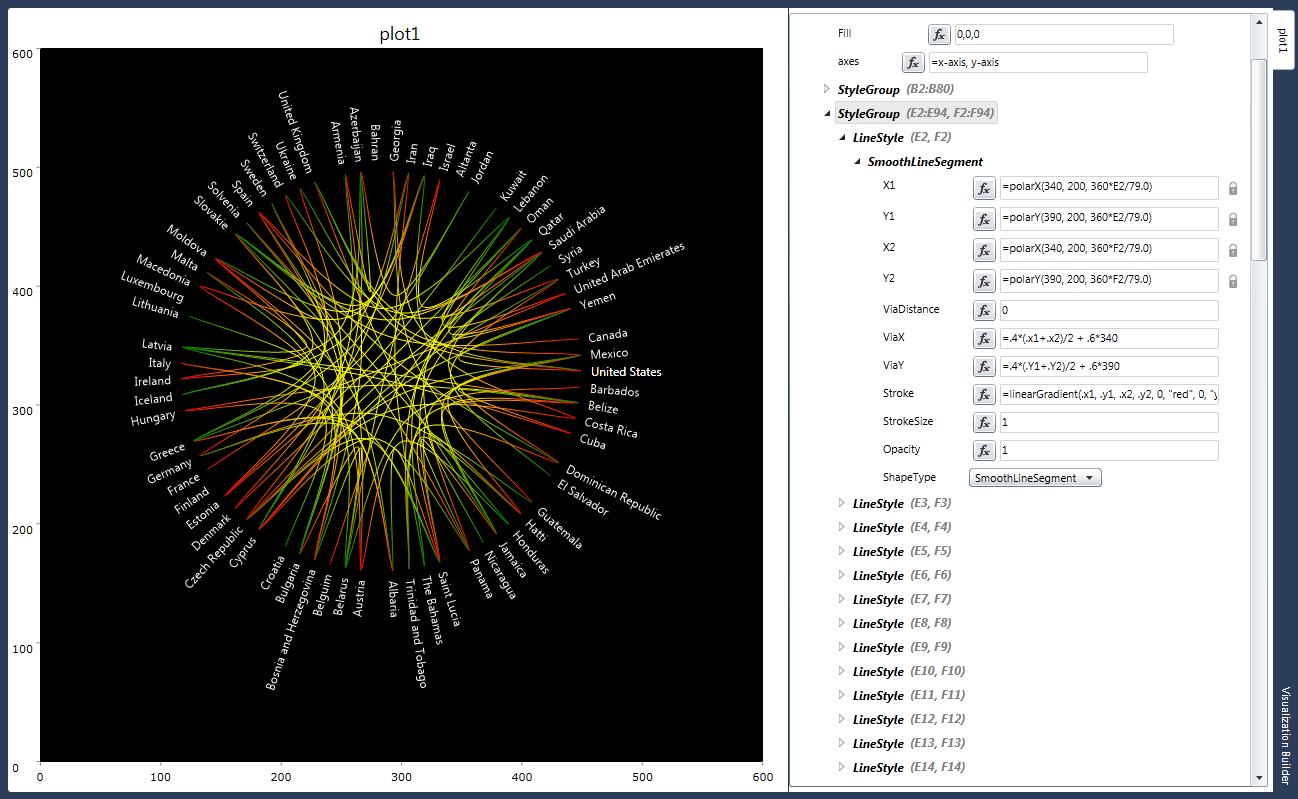
 

Figure 9 (left). A hi-lo stock chart showing 40 months of data for a single stock listing. Each bar represents the high, low, opening, and closing values for a single trading month as three boxes. Months that increase are drawn in green, while decreases are in red. Non-contiguous pairs occur from after-hours trading.   
(right) A radial visualization that shows connection between countries. This was done by combining two visual styles: one for the text labels, and the other for the arcs between nodes.

Many of the users commented that while they understood the system, their preferred working style would be to start from an already existing visualization and modify that in order to get the desired visualization - they did not think about starting with a blank canvas, adding a visual element to the page and subsequently binding data to that element. Several users felt that the ability to examine an existing visual style was appealing, though at least one user (the programmer) felt that he would have been more effective using raw code.

The study already prompted some redesigns of the tool. Initially, following the example of tools like Microsoft Word and Adobe Illustrator, changing a single element in a style does not update all the others; rather, the user needs to manually press “update style.” We initially felt that it made more sense for users to change a stable chart, and then propagate the changes. Several users found this extra step annoying, and asked for all marks to be changed together when updating a single mark.

Similarly, Vis-à-vis currently represents every mark on the canvas in a hierarchical list. This could be overwhelming when dealing with many rows of data. It may be more intuitive to represent the global style template, and then highlighting items that have been explicitly modified.

# Discussion & Future Work

While the results of the prototype and the evaluation are promising, the evaluation helped highlight several areas for future work. All of the participants in the study reported that they prefer to pick from a gallery of existing templates for starting their visualizations. While users were able to adapt a template for their own purposes, the design must ensure that this is well supported. Can users find the visualization that is close enough to adapt for their own purposes? How easy is it for them to understand the visual style in order to change it? As the system can quickly update to match changes to a visual style, users can experiment and see their results immediately. None the less, while the user study above validates the basic design, further testing is needed to establish whether it is sufficient.

As we noted above, users wanted the interface to emphasize scaffolding previous charts. It would be desirable to improve the sharing interface to help show how charts have been modified from previous ones, and are building a larger core of starting visualizations.

Currently, the prototype allows a very limited form of sharing: styles can be exported and brought into other spreadsheets. This design could be easily extended to include visualization-sharing features, integrated into the tool, where users can upload, download, browse, and modify each other’s styles.

## Limitations

Vis-à-vis assumes a one-to-one relationship between data items and marks or components of marks. It is able to handle polylines in which each control point maps to a data point. However, many charts require additional data columns that are implicit rather than explicit in the spreadsheet. The rectangles in a treemap, or the buckets in a histogram, cannot be computed in the fields of the objects; we need an auxiliary column to compute these additional fields. These additional columns should be associated with styles, rather than with the data, in order to facilitate sharing of visualizations. We plan to develop an infrastructure for styles to carry additional columns which can be addressed within shape fields.

# Conclusions

While information workers may be satisfied with a small set of visualizations, there is often the need to use visualizations that are not built into a current system. In Vis-à-vis, we allow customization and extensibility within a single framework, using the same working paradigm that knowledge workers are already familiar. By allowing end-users to explore the specifications for existing visualizations, we have strived to create a virtuous cycle: most users can use and customize a large variety of initial visualizations and some workers can extend and create new visualizations that are available for the community as a whole.

# References

[ABE09] Abraham, R., Burnett, M., Erwig, M. 2009. *Spreadsheet Programming*. Wiley Encyclopedia of Computer Science and Engineering. 2804–2810.

[BH09] Bostock, M., Heer, J. *Protovis: A Graphical Toolkit for Visualization*. Transactions on Visualization and Computer Graphics 2009.

[CRB\*98] Ed H. Chi, John Riedl, Phillip Barry, Joseph Konstan. *Principles for Information Visualization Spreadsheets*. IEEE Computer Graphics and Applications (Special Issue on Visualization), pp. 30--38. July/August, 1998. IEEE CS Press.

**[**CRK97] Chuah, M. C., Roth, S. F., and Kerpedjiev, S. 1997.*Sketching, Searching, and Customizing Visualizations: A Content-based Approach to Design Retrieval***.** In M. Meybury (ed.), Intelligent Multimedia Information Retrieval. AAAI/MIT, 83-111.

[Fek04] Fekete, J.-D. 2004. *The InfoVis Toolkit*. Proceedings of the 10th IEEE Symposium on Information Visualization (InfoVis'04), IEEE Press, pp. 167-174.

[FDF\*10] Fisher, D., Drucker, S., Fernandez, R., Ruble, S. 2010. *WebCharts: Extending Applications with Web-Authored, Embeddable Visualizations*. IEEE Transactions on Visualization and Computer Graphics. 16(6, Nov/Dec), 1325-1332.

[HCL05] Heer, J., Card, S. K., Landay, J. 2005. *Prefuse: A Toolkit for Interactive Information Visualization*. ACM Human Factors in Computing Systems (CHI 05). April. 421-430.

[IT09] Itoh, M., Tanaka, Y. 2009. *A Framework of Constructing Coordinated Multiple 3D Visualizations in Excel*. Proceedings of IEEE Information Visualization (IV). Barcelona. 162-170.

[Lev94] M. Levoy, 1994. *Spreadsheets for Images*. Proceedings of Siggraph 1994, A. Glassner, ed., ACM Press, New York, , pp. 139-146.

[Lew90] Lewis, C. 1990. *NoPumpG: Creating interactive graphics with spreadsheet machinery*. In Visual Programming Environments: Paradigms and Systems, E. Glinert, Ed. IEEE Computer Society Press, Los Alamitos, Calif., 526-546.

[Mac86] Mackinlay, J. 1986. *Automating the Design of Graphical Presentations of Relational Information*. ACM Transactions on Graphics. 5 (2, April). 110-141.

[Pie86] Piersol, K. 1986. *Object-Oriented Spreadsheets: The Analytic Spreadsheet Package*, Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (Sigplan Notices, Vol. 21, No. 11), N. Meyrowitz, ed., ACM Press, New York, pp. 385-390.

[RC94] R. Rao and S.K. Card. 1994. *The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus + Context Visualization for Tabular Information*, Proceedings of the Conference on Human Factors in Computing Systems (CHI ‘94), Boston, MA, April, pp. 318-322

[RKM\*94] Roth, S.F., Kolojejchick, J., Mattis, J., and Goldstein, J. 1994. *Interactive Graphic Design Using Automatic Presentation Knowledge.* Proceedings of the Conference on Human Factors in Computing Systems (CHI '94), Boston, MA, April, pp. 112-117.

[SS09] Stewart, R. and schraelfel, m*.* 2009. *Graph Sketcher: Extending Illustration to Quantitative Graphs.* Proceedings of the Conference on Human Factors in Computing Systems (CHI ‘09), Boston, MA, April, 1113-1116*.*

[Wea04] Weaver, C. 2004. *Building Highly-Coordinated Visualizations In Improvise*. Proceedings of the IEEE Symposium on Information Visualization, Austin, TX, October 2004.

[Wic09] Wickham, H. 2009. *ggplot2: elegant graphics for data analysis.* Springer*.*

[Wil99] Wilkinson, L*.* 1999. *The Grammar of Graphics.* Springer.