Volume xx (200y), Number z, pp. 1–13

Visualization Mosaics for Multivariate Visual Exploration

S. MacNeil[1] and N. Elmqvist[†2]

[1]Purdue University, USA

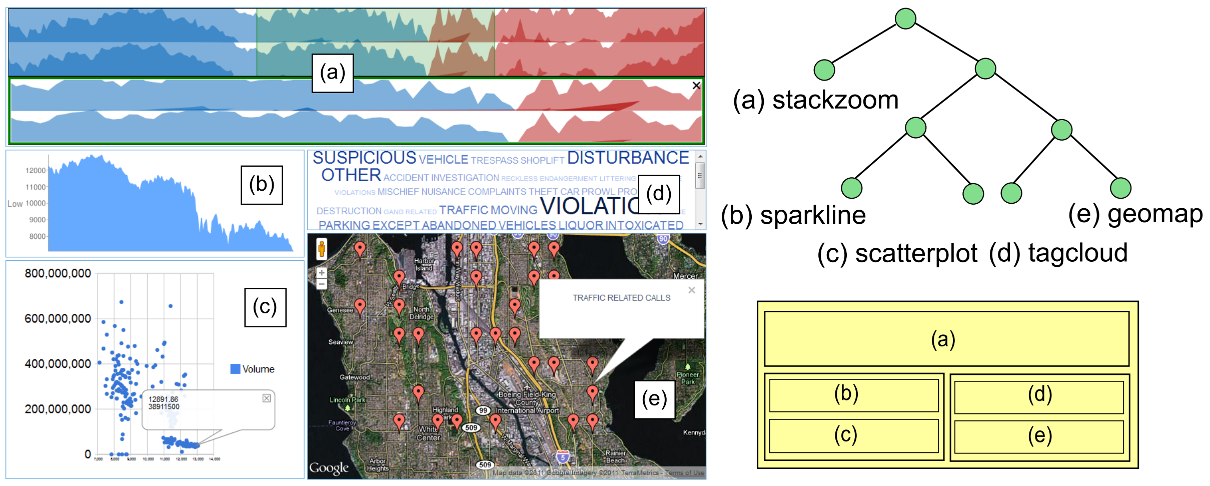


Figure 1: *MosaicJS—our visualization mosaics implementation—visualizing a dynamic police incident database (left) for the city of Seattle (data source: http://data.seattle.gov/). The tile hierarchy of the mosaic is given to the right.*

Abstract

We present a new model for creating composite visualizations of multidimensional datasets using simple visual representations such as point charts, scatterplots, and parallel coordinates as components. Each visual representation is contained in a tile, and the tiles are arranged in a mosaic of views using a space-filling slice-and-dice layout. Tiles can be created, resized, split, or merged using a versatile set of interaction techniques, and the visual representation of individual tiles can also be dynamically changed to another representation. Because each tile is self-contained and independent, it can be implemented in any programming language, on any platform, and using any visual representation. We also propose a formalism for expressing visualization mosaics. A web-based implementation called MosaicJS supporting multidimensional visual exploration showcases the versatility of the concept and illustrates how it can be used to integrate visualization components provided by different toolkits.

Categories and Subject Descriptors (according to ACM CCS): H.5.2 [Information Interfaces and Presentation]: User Interfaces—Interaction styles I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1. Introduction

The field of visualization is changing. The traditional visualization pipeline [CMS99] is being transformed into a shorter, more flexible pipeline [McK09] that is driven by the endless capabilities of the Web 2.0 ecosystem. Visualization is

[†] School of Electrical & Computer Engineering, Purdue University, 465 Northwestern Ave, West Lafayette, IN 47907, USA, E-mail: elm@purdue.edu.

becoming an increasingly social activity [HVW07] targeted towards the masses [VWvH*07]. Complex visual representations that encode dozens of individual data dimensions in a single view (e.g., [Ins85, Kei00]) are making way for composites [JE12] of small and simple views based on classic statistical graphics [CM84, TU08], such as information dashboards [Few06], Polaris [STH02], Dashiki [McK09], ScatterDice [EST08], and Jigsaw [SGL08]. In other words, monoliths are becoming mosaics: multiple simple views with mutual coordination [Rob07], instead of heavyweight monolithic representations.

The key feature in this new generation of visualization systems is not the ability to represent all data in a single view, but rather the capability for effortless creation of new views, and for coordinating all of these views to support comparison and correlation. Just like an architect or engineer would not stop at a single blueprint when designing a building or a complex piece of machinery, information analysts—causal or expert alike—need to create **several** views, each one portraying a unique aspect of the data. However, existing information dashboards [Few06] are typically static and thus poorly support this quick and fluid [EMJ*11] visualization process. Those that do support dynamic exploration and reconfiguration, such as Google's web-based dashboards, typically do not make **relations** between different views explicit, which is vital for understanding the overall data context.

In this paper, we propose a new theoretical model for coordinated multiple views (CMV) [Rob07] of multidimensional data that builds on this trend and that we therefore call *visualization mosaics*. A visualization mosaic is a hierarchical container structure for visual representations (*tiles*) that visualize a particular dataset. Unlike traditional dashboards, the mosaic is dynamic and allows rearranging, repurposing, or restructuring the component tiles on the fly. A key aspect here is that the tiles are treated as black boxes, and can thus be any external visualization component provided by a third party. The main features of mosaics are the following:

- **Space-filling layout:** a slice-and-dice graphical layout that conveys the hierarchical structure of the mosaic;
- **Visual transformations:** a grammar for transforming between different visual representations (equivalent or not);
- **Mosaic management:** interactions for splitting, merging, arranging, and transforming tiles on the mosaic; and
- **Formal notation:** a language for specifying the structure and contents of a visualization mosaic.

The metaphor employed by the visualization mosaic concept is one of actually building a mosaic of views by splitting, merging, +and arranging tiles on the visual canvas using a pre-defined set of simple visual representations for multidimensional, temporal, and spatial data. The mosaic, which is the final product of an analysis session, is essentially an interactive visualization dashboard [Few06] consisting of a collection of information tiles, i.e. mutually linked and coordinated views. Figure 1 shows an example mosaic for a real-time crime dataset. The tile layout on the visual canvas exposes the underlying structure of the dataset.

Our intended use for the visualization mosaic concept is primarily for web-based data analysis, similar to Dashiki [McK09], ManyEyes [VWvH*07], and VisGets [DCCW08]. Towards this end, we have implemented a prototype implementation of the concept as a JavaScript framework that can run in any modern web browser. The framework uses the Google Data Source API for data management, enabling us to use virtually any web-based visualization toolkit—including the Google Visualization API, Protovis [BH09], and D3 [BOH11]—as component tiles in a mosaic. Using this prototype, we give two case studies for how the mosaic concept could be used in practice.

2. Related Work

This work represents the merging of several different topics within visualization and visual analytics research:

- **Statistical data graphics:** the work originates from a long tradition of graphical representations of data;
- **Multidimensional visualization:** the emphasis is primarily on multivariate datasets;
- **Coordinated multiple views:** the approach to visual exploration [Kei02] is based on multiple linked views;
- **Visualization dashboards:** the end result is a visualization dashboard (but a dynamic one); and
- **Web-based visualization:** the target audience is the Internet and our target platform is the web browser.

2.1. Statistical Data Graphics

Graphical representations have long helped statisticians make sense of their data [Fri07]. The Scottish engineer William Playfair (1759–1823) is hailed as the father of such statistical graphics [**?**], and used line, bar, pie, and circle graphs to communicate political and economical data to his fellow citizens and policymakers in Georgian era England. Data graphics have progressed much since then and are used for many disciplines of mathematics, as well as domains such as finance, politics, science, engineering, and medicine.

In particular, combining multiple graphs of data to reinforce a message dates back more than 25 years [BC87, BCS96], and has often been combined with *linked highlighting* between the graphs. One of the early examples was Paul Velleman's DataDesk software (`http://www.datadesk.com`), introduced already in 1985, which supported a graphical user interface, linked displays, and 3D plots on a standard personal computer, as well as John Tukey's PRIM-9 [TFF88] system for exploratory data analysis from the late 1980s. Today's statistical packages such as SAS, SPSS, and R (`http://www.r-project.org`) all support multiple views in various forms.

2.2. Multidimensional Visualization

Multidimensional data is one of the classic information visualization datasets [Shn96], and much research has been devoted to such data over the years. As opposed to the largely spatial 2D or 3D data used in scientific visualization, multidimensional datasets are those that contain more (sometimes many more) dimensions than three. Classic multidimensional visualization techniques include scatterplot matrices [CM84], parallel coordinates [Ins85], dense pixel displays [KK94], and stacked displays [LWW90].

Many systems for multidimensional visual exploration have been built, starting with Tukey's original PRIM-9 [TFF88] system, Becker and Cleveland's trellis displays [BCS96], Ward's XmdvTool [War94], and the XGobi/GGobi systems [SLBC03]. Several commercial visualization systems now also exist, such as Spotfire [Ahl96] and Tableau (formerly Polaris [STH02]). All of these promote an open-ended visual process for exploring datasets.

The Hierarchical Visualisation Expression notation [SDW09] by Slingsby et al. is interesting because it provides a formal notation for describing hierarchical layouts of multidimensional datasets, resulting in reconfigurable space-filling visualizations of small multiples (conditioned by the layout) of this data. Our mosaic framework and notation is similar, but is designed for interactive visual exploration and integrates multiple different visualization types into the resulting mosaic instead of small multiples using the same visual representation.

Several other recent academic offerings also exist. XmdvTool [War94] continues to be developed and now supports a rich plethora of techniques—recent extensions include nugget management [YRW07] and data quality support [CWRY06]. Improvise [Wea04], also discussed below, promotes a highly extensible and configurable approach to building multidimensional visualizations consisting of many cross-linked views. Dust and magnets [YMSJ05] uses interaction and animation to support exploration and insight. The DataMeadow [EST08] is an interactive environment with a visual query language. Finally, ScatterDice supports multidimensional analysis using scatterplot matrix navigation [EDF08], where users can explore and query data from different views connected by animated transitions.

2.3. Coordinated Multiple Views

Adding multiple views of data has its roots in linked graphs from more than 25 years of statistics [BC87, BCS96], and has often been combined with the notion of *linked highlighting*. It is also a common strategy in interactive visualization for dealing with multidimensional datasets [BWK00]; examples of this practice include Mondrian [TU08], Jigsaw [SGL08], Dashiki [McK09], Improvise [Wea04], and Tableau/Polaris [STH02]. This practice is coupled with the proliferation of multi-monitor setups that increase the available display space on personal computers and enable space management to support analysis [AEN10, Gru01].

However, multiple views can be overwhelming without a clear organizing principle. The most common approach to organize multiple views is called *coordinated multiple views* (CMV) and simply juxtaposes views in the same visual space with *brushing* [BC87]—dynamic highlighting of items selected in one view in all other views—as the main coordination mechanism. Baldonado et al. [BWK00] proposed a set of guidelines for designing CMV systems in 2000; more recent work in this area includes Roberts state-of-the-art report from 2007 [Rob07], and Javed and Elmqvist's more general characterization of the design space of composite visualizations [JE12]. While there exists many CMV systems (and, consequently, while many visualization systems use CMV techniques), Improvise [Wea04] is perhaps the most well-known of these. Essentially a visualization toolkit, Improvise is designed for building multi-view visualizations that are mutually linked and brushed. However, to our knowledge, other than the pure toolkits reviewed below, there currently exists no full-fledged CMV systems designed for web-based settings.

The prior art most relevant to our work is snap-together visualizations [NS00, NCIS02], which allow users to rapidly and dynamically mix visualizations and coordination mechanisms using direct manipulation. Coordination builds on the relational data model, where data tables are loaded into visualizations and a join relationship is defined between linked tables. Our approach is similar, but uses a single table that is partitioned into a tile hierarchy, enabling us to explicitly represent relations between visualizations.

2.4. Visualization Dashboards

A visualization (or information) *dashboard* is a graphical canvas consisting of multiple visual representations with live connections to data sources [Few06]. They are typically assembled for a specific purpose. Essentially inspired by physical in-vehicle dashboards, an information dashboard is designed for quick, almost preattentive, survey of a set of parameters and data values; in other words, the emphasis is on quick reference rather than in-depth analysis. Information dashboards are becoming popular for domains such as business intelligence, and some commercial dashboard systems include SAP Xcelsius and Spotfire Posters.

Despite their popularity, traditional information dashboards have been criticized for being static, opaque, and difficult to build and maintain for all but expert users [Few06, McK09]. These weaknesses serve as motivating factors for continued work on more dynamic and configurable information dashboards. Perhaps the most recent and relevant of these systems is Dashiki [McK09] (publicly known as Many Eyes Wikified), which combines a user-edited Wiki with a markup syntax that allows the user to lay out visualizations from the Many Eyes [VWvH*07] site side-by-side. This supports a dynamic approach to composing multiple visualizations that have live connections to their data sources, and combining them with text and hyperlinks.

The recent WebCharts [FDFR10] framework encapsulates visualization components implemented in different languages using a common interface that enables these components to be dynamically added to any application supporting the interface. Our visualization mosaics use the same idea, yet take a dashboard approach to presenting the components.

Finally, in very recent work, Lex et al. [LSS*11] propose VisBricks, a framework for constructing what are essentially dashboards consisting of multiple component visualizations. The framework is designed for large and inhomogeneous datasets, similar to our visualization mosaics concept. However, unlike our work, the VisBricks system is a native application based on OpenGL and thus has higher rendering and data management performace, but requires installation, specific libraries, and cannot easily integrate visualization components created by third parties on the Internet.

2.5. Web-based Visualization

The Web is the great enabler for the rise of visualization for the masses, a trend that was initiated by online visualization websites such as NameVoyager [Wat05] and Many Eyes [VWvH*07]. While traditional visualization software must often be built for a specific operating system and computing platform, the web browser has the lure of a truly cross-platform target—at least in theory.

VisGets [DCCW08] was one of the first web-based visualization systems that made use of multiple coordinated views to show temporal, spatial, and semantic data about a web-based query. The visualization tiles in our mosaic approach can be seen as VisGets with automatic mutual linking. Dörk et al. continue on this line of inquiry in a very recent paper on a Visual Backchannel [DGWC10] system for the Web where multiple coordinated views allow the user to follow Twitter posts about a specific event in real time. However, the VisGets and Visual Backchannel systems are specifically designed for particular applications and are not easily customizable for general datasets.

Several general APIs have been proposed for web-based visualization. Protovis [BH09] is a general SVG-based JavaScript library for building visualizations. The more D3 [BOH11] uses the actual document object model to visualize data. The JavaScript InfoVis Toolkit (`http://thejit.org`) is another JavaScript-based visualization library. Finally, the Google Visualization API (`http://code.google.com/apis/chart/`) provides a general data source interface that enables visualization components to be written in any language while still sharing data. While all of these APIs are general enough to enable building mosaics similar to our work, they require programming to deploy. Visualizations built in Protovis, D3, and Google Vis can still be used as components in our system, however. Instead of requiring programming, we are more concerned with merging component visualizations into a coherent whole during interactive exploration.

3. Visualization Mosaics

A *visualization mosaic* is a composite visualization consisting of multiple visual representations juxtaposed in the same visual space with each representation visualizing a subset of a master dataset. Mosaics are examples of coordinated multiple view (CMV) [Rob07] visualizations, but have stricter data and layout constraints. These restrictions allows us to completely specify a mosaic using a formal notation. In the below section, we first give a model and some basic definitions, present the notation, and show how this gives rise to a natural hierarchy-preserving slice-and-dice layout.

3.1. Model

Our visualization mosaics build on the data model of Card et al. [CMS99] where a dataset D is a data table consisting of dimensions and cases. Using the convention set forth by Fekete [Fek04] as well as Heer and Agrawala [HA06], we specify the table in terms of a *data column* C, i.e., $D = \{C_1, \dots, C_n\}$. A column is an abstraction of a dimension in a data table and contains one value per item in the table (accessible using the row number). It also has a collection of meta-data such as name, type, value range, average, etc. Retrieving a specific row in the table amounts to retrieving the value for that row from every column in the table.

Visualization mosaics are defined for a master dataset that is common among all of the individual visual representations in the mosaic. We use the row number to uniquely identify the same item across all parts of the mosaic, such as for brushing [BC87]. This means that all columns must belong to the same table. In other words, if we want to visualize more than one dataset, we must first combine the datasets into a single dataset, for example using a join operation.

Given the above definitions, a visualization can be seen as a function V that accepts columns as input and creates a graphical representation based on visualization technique as output. For example, $V_{scatter}(C_1, C_2)$ would denote a 2D scatterplot of two columns C_1 and C_2 from some dataset.

3.2. Notation

We use the following grammar for defining our notation, where a $< mosaic >$ element is the top-level expression:

$$\begin{aligned}
< mosaic > &::= < group > \mid < tile > \\
< group > &::= \texttt{G(} < mosaic > + \texttt{)} \\
< tile > &::= \texttt{T(} < type >, C_n + \texttt{)} \\
< type > &::= \emptyset \mid \texttt{scatter} \mid \texttt{parcoord} \mid \texttt{hist} \ldots
\end{aligned}$$

Here, $< type >$ denotes visualization types that a tile can use, and this element is only limited to the visual representations supported by the actual mosaic implementation.

Given the above grammar for our mosaic notation, here is an example mosaic M_1 consisting of two scatterplots that each use two columns of the dataset (Figure 2(a)):

$$M_1 = \texttt{G(T(scatter,}C_1\texttt{,}C_2\texttt{),T(scatter,}C_3\texttt{,}C_4\texttt{))} \quad (1)$$

The real estate mosaic in Figure 2(b) consists of a geographical map where the houses are plotted using their longtitude and latitude (with their address as labels). In addition, we also create a histograms for the price, square footage, and acreage of the houses in the real estate database.

$$\begin{aligned} M_2 = \; & \texttt{G(T(map2d,}C_{lat}\texttt{,}C_{long}\texttt{,}C_{addr}\texttt{),} \\ & \texttt{G(T(hist,}C_{price}\texttt{),T(hist,}C_{sqft}\texttt{),} \\ & \texttt{T(hist,}C_{acres}\texttt{)))} \end{aligned} \quad (2)$$

Each house in the above mosaic is represented by an item in a dataset $D_{houses} = \{C_{addr}, C_{lat}, C_{long}, C_{price}, C_{sqft}, C_{acres}\}$. This allows a mosaic implementation to easily support brushing by simply adding a meta-column [Fek04] $C_{\#selected}$ of booleans that all visualizations access in order to determine which items to highlight (which has been shown to be an efficient approach by Fekete and Plaisant [FP02]).

3.3. Constructing Mosaics

To facilitate building mosaic expressions, we organize the columns in a dataset $C_n \in D$ into a column hierarchy using the above mosaic structure. In other words, the starting point of a mosaic is a blank visual space, $M = \texttt{T}(\emptyset, D)$, where none of the columns in the dataset are allocated to a visualization.

Constructing a mosaic from this empty space is primarily done using two operations: split and merge. These are used to add or remove tiles or groups from the mosaic hierarchy:

- **Split:** Split a tile of `<type>` t into a group of n tiles, each with their own type t_i and subset D_i of the dataset D: $\texttt{T}(t,D) \Rightarrow \texttt{G}(\texttt{T}(t_1, D_1 \subset D), ..., \texttt{T}(t_n, D_n \subset D))$.
- **Merge:** Merge a group containing one or several tiles, each with their own `<type>` t_i and dataset D_i, into a single tile with type t and dataset D as the union of all D_i: $\texttt{G}(\texttt{T}(t_1,D_1),...,\texttt{T}(t_n,D_n)) \Rightarrow \texttt{T}(t,D)$ where $D = \bigcup_{i=1}^{n} D_i$.

Exploratory data analysis [Kei02, Tuk77] using a visualization mosaic is a progressive refinement [GZA06] of a dataset using a series of split and merge operations as the user explores the data, studies correlations between dimensions, and creates and destroys tiles. The final mosaic expression is a representation of the exploration process, but is still dynamic and can be refined further at a later date. In fact, by storing the sequence and parameters of split and merge operations, it is easy to support undo and redo in the exploration(e.g. [KAF*08]). Furthermore, the mosaic expression captures its visual layout (see Section 3.5).

3.4. Visual Representations

The initial visual representation of a new tile is the `empty` visual representation, i.e., an empty space. To improve on this, an actual implementation may choose to implement logic to deduce a suitable visual representation given a set of columns with their names and data types, similar to the work by Mackinlay et al. [Mac86, MHS07].

Tiles are containers for both (a) visualizations, which are given a specific area inside the tile's layout allocation to render themselves on, and (b) subsets D_i of the columns of the mosaic's master dataset D. A visualization maps one or several of the data columns in D_i to visual variables on the graphical rendering of the visualization. Thus, changing the visualization type of a tile means that the visual representation contained inside the tile is replaced, and the mapping from data columns to visual variables will also change.

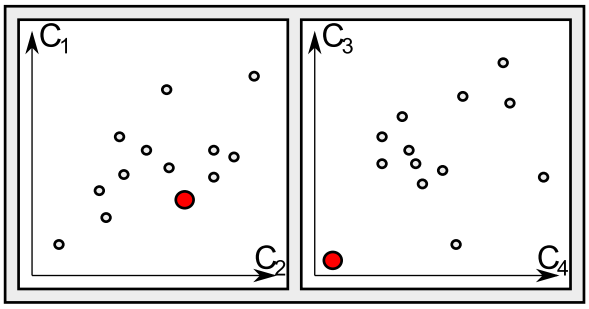
To make the change in representation clearly visible, one solution may be to use an *animated transition* where the transformation from one visual representation to another is shown using a smooth animation (similar to those of Heer and Robertson [HVW07]). Item identities are preserved across all columns in the mosaic, which makes it possible to morph visual marks in one representation to the marks in another. However, transitions must still be specifically designed between each pair of visual representations.

3.5. Layout

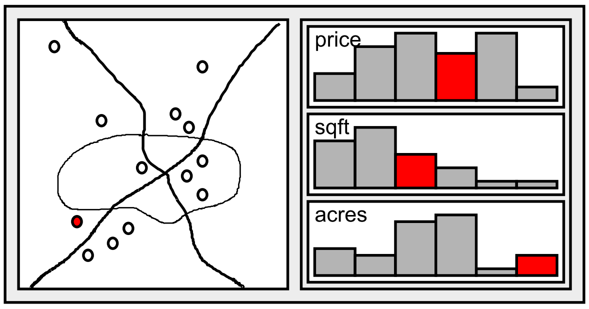
Our objective with this work is that a mosaic M and an dataset D should completely describe the appearance of the mosaic on the screen. To achieve this, we utilize the hierarchy of views in a mosaic as input for a space-filling slice-and-dice layout inspired by treemaps [Shn92], and previously used for view layout in SimVis [DGH03]. This algorithm recurisvely splits the available space between children in an internal node (group) depending on their number of data dimensions. Following treemap convention, the slices alternate between horizontal and vertical for each step.

There are two benefits of this approach: (1) a mosaic expression completely encodes specifies the visual appearance of the resulting mosaic, achieving a compact representation (which is significant in web-based system); and (2) the layout communicates the structure of the mosaic to the user (i.e., how the dataset has been subdivided and drilled into). For example, Figure 2(a) shows the mosaic layout of the two scatterplots in Equation 1, and Figure 2(b) shows the mosaic for Equation 2. The nesting gives an indication of the visual exploration process. However, we also allow the user to drag tile borders to resize their layout allocations as necessary.

To further improve the layout algorithm, we also introduce the notion of *aspect ratio affinity*. Different visual representations have different optimal aspect ratios: for example, a parallel coordinate plot works best for wide aspect ra-



(a) Mosaic with two scatterplots side by side.



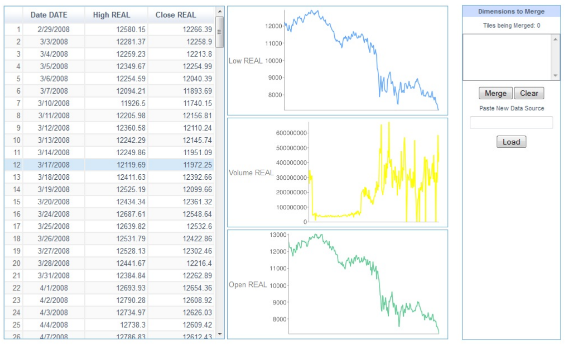
(b) Mosaic with a geographical map and histograms.

Figure 2: *Two example visualization mosaics for Equations 1 and 2, respectively.*

tios, whereas a scatterplot yields best results for a square aspect ratio. By taking aspect ratio affinity into account when computing the layout, the dashboard appearance can be optimized depending on the component visualizations. In other words, instead of blindly alternating between vertical and horizontal slicing, the layout algorithm chooses which direction to slice depending on available space for each node.

4. Implementation: MosaicJS

To validate our new model for coordinated multiple views, we built a web-based implementation of visualization mosaics in JavaScript called MOSAICJS (Figure 3). By using a web-based solution, we were able to make our implementation work across platforms, operating systems, and browsers. This approach also allows us to harness existing web-based visualization toolkits such as the Google Visualization API, RaphaelJS, and Protovis [BH09] in our implementation.



	Date DATE	High REAL	Close REAL
1	2/29/2008	12580.15	12266.39
2	3/3/2008	12281.37	12258.9
3	3/4/2008	12259.23	12213.8
4	3/5/2008	12349.67	12254.99
5	3/6/2008	12254.59	12040.39
6	3/7/2008	12094.21	11893.69
7	3/10/2008	11926.5	11740.15
8	3/11/2008	12205.98	12156.81
9	3/12/2008	12360.58	12110.24
10	3/13/2008	12242.29	12145.74
11	3/14/2008	12249.86	11951.09
12	3/17/2008	12119.69	11972.25
13	3/18/2008	12411.63	12392.66
14	3/19/2008	12525.19	12099.66
15	3/20/2008	12434.34	12361.32
16	3/24/2008	12687.61	12548.64
17	3/25/2008	12639.82	12532.6
18	3/26/2008	12531.79	12422.86
19	3/27/2008	12528.13	12302.46
20	3/28/2008	12441.67	12216.4
21	3/31/2008	12384.84	12262.89
22	4/1/2008	12693.93	12654.36
23	4/2/2008	12790.28	12608.92
24	4/3/2008	12734.97	12626.03
25	4/4/2008	12738.3	12609.42
26	4/7/2008	12786.83	12612.43

Figure 3: *MosaicJS visualizing stock market data from 1986 to 1997. The tiles use Google Visualization API components. The interface components on the right are used to merge existing tiles; splitting and subdividing tiles is done in the "Data" tab of the tile the operation will be performed upon.*

The general architecture of MosaicJS is built around a single data table that represents the master dataset in the visualization mosaic formalism. The framework also maintains a mosaic hierarchy where this master table is subdivided into many smaller tables (data views) that are visualized independently as tiles. This mosaic hierarchy is used for organizing, positioning, and sizing tiles on the display. Each tile is responsible for only visualizing the data table that it contains, and its visual representation can be configured by the user.

4.1. Data Access

MosaicJS uses the Google Data Source API for accessing and reading data from data sources on the web; examples include spreadsheets, RSS feeds, static webpages, etc. The core data table structure used in our framework is the one that is defined by the Google Data Source API. This has the added benefit of making the data more dynamic by allowing changes to the data to be reflected in the visualization.

4.2. Layout and Control

Our MosaicJS framework maintains a mosaic hierarchy in the form of a tree consisting of groups and tiles (similar to the mosaic grammar in Section 3.2). As before, tiles are leaf containers that hold visual representations, whereas groups are internal nodes in the mosaic hierarchy. We use the slice-and-dice algorithm described in Section 3.5 to organize components on the visual space. Our implementation assigns space to children proportionally to the number of columns they contain relative to their parents.

The mosaic hierarchy can be changed in two ways: either by **splitting** a tile into smaller tiles, which represent a subset of their parent tile's data table, or by **merging** several tiles, which effectively merges all of their dimensions into one table. When merging tiles, first the LCA (Lowest Common Ancestor) must be found. It is this ancestor in which the two data tables are merged with the existing table in that tile.

When a tile is created, it is a container for a subset of its parent's columns. It has a position and a dimension on the visual space, but no visual representation that maps its columns to graphical shapes. MosaicJS renders a tile as a floating HTML5 element that contains additional elements for menu bars and tabbed views controlled by the user:

- **Vis:** The graphical view of the visual representation chosen for the tile (initially empty);
- **Data:** A list of the columns contained within the tile as well as the split and subdivide operations;
- **Options:** Selection and configuration of the tile's visual representation and its visual mappings.

4.3. Visual Representations

Because MosaicJS completely insulates the contents of each tile from global layout and data management, the framework does not impose any particular requirements on the visual representations used. Any visual component that can be contained within an HTML floating element can be used inside MosaicJS, but doing so requires writing a JavaScript wrapper between the component and our framework.

Our current implementation supports visual representations such as point charts, scatter plots, and parallel coordinates. We have also implemented a Google map view that renders geotagged data, similar to Jusufi et al. [JJKM08] and Ho et al. [HLAJ11]. Furthermore, we also support visualization examples from D3 [BOH11] as well as a web-based timeline visualization with stack zooming [JE10].

4.4. Coordination

Because the visual representations in each tile are entirely independent (they may even be implemented in entirely different programming languages and using different libraries), coordination between tiles is challenging. Accordingly, MosaicJS contains two basic forms of coordination:

- **Data coordination:** The MosaicJS framework splits the master dataset into subtables for each tile. This is the most basic form of coordination in the framework.
- **Selecting and brushing:** All tiles that use a Google Visualization API component may optionally implement the methods `getSelection()` and `setSelection()`, which use an array of indices to communicate which rows in a table should be selected (and thus highlighted). MosaicJS uses these methods to propagate selections in one tile to all other tiles, thereby achieving brushing [BC87]. Note that not all visualizations are required to implement this; certainly not those that are not GVis components. Future improvements to MosaicJS would establish a standardized interface for brushing across all components.

4.5. Interaction

Beyond selecting and brushing, the primary interactions in MosaicJS are accessed either using the tabbed views on a particular tile, using the control panel interface (right side of Figure 3), or by clicking directly on the visual representation. The following main interactions are supported:

- **Subdivide:** Split the selected columns in the current tile into a sibling tile with the same parent as the original.
- **Split:** Split the selected columns in the current tile into a child tile with those columns.
- **Merge:** Merge the selected tiles into a single tile. This requires finding the least common ancestor (LCA) of those tiles, which will become the new tile.
- **Select:** Select a single point in a visual representation in one tile, which will highlight this point in all of the other visual representations. However, this requires a shared meta-column between all visualizations that mark whether a point is highlighted or not, and not all third-party visual components support this feature.

4.6. Importing and Exporting Data

One of the primary benefits of a web-based visualization system is that it can be integrated into the overall Web 2.0 ecosystem of data sources, RSS and Atom feeds, and user contribution mechanisms [McK09]. MosaicJS is based on the Google Data Source API, which also allows it to leverage importing data from spreadsheets, CSV files, and Atom and RSS feeds. Furthermore, we provide data conversion wrappers that accept JSON and JavaScript data structures as input. While a MosaicJS dashboard will not automatically update as dynamic data comes in, it is possible to manually refresh a dashboard at any time: this will keep the data flow structure and visual layout of the mosaic unchanged, but will repopulate the views with the new data.

Exporting the results of an analysis is also an interesting aspect of web-based visualization. The end result of a visual exploration session using MosaicJS is a dynamic visualization dashboard that should be possible to share with other, perhaps less knowledgeable, users. While MosaicJS is currently a prototype, it is easy to envision a production implementation where the mosaics technique is integrated in a social data analysis website similar to ManyEyes [VWvH*07] that supports comments, annotation, and sharing of mosaics. The current implementation only supports exporting the state of a mosaic by serializing to XML, or by capturing a bitmap image of its visual appearance.

4.7. Implementation and Performance

Our implementation is built in JavaScript and HTML5, and uses the Google Data Source API for data access. The key component is the layout manager, which supports the slice-and-dice layout and pipes data dimensions to different tiles. The toolkit currently supports components from the Google Visualization API and D3 [BOH11]; adding new components is simply a matter of writing the appropriate JavaScript wrapper and transforming data from the Google Data Source API into a format the component visualization can handle.

Performance is another issue for web-based applications that rely on the browser as the run-time environment. Visualizations are typically rendering-intensive, which traditionally has been an issue in web browsers (although recent developments in browser technology indicate that this state of affairs is improving [BH09, BOH11]). However, since MosaicJS only connects existing visualization components and does not provide any visual representation on its own, its performance is largely dependent on the performance of the external visualization tiles themselves. For this reason, we have not run performance testing, but our informal observations is that the mosaics architecture is scalable and efficient.

5. Examples

We present two examples of how to use our MosaicJS implementation: for 911 police incidents, and for rock climbing.

5.1. Seattle Crime Analysis

Crime is a ubiquitous problem in society. Establishing a safe environment for a city's citizens is essential to a city's development. However, controlling crime is a significant challenge for law enforcement agencies, which are struggling to manage increasing crime with decreasing money and personnel. Determining which areas are crime hot spots can mitigate peoples' risk of being victimized, especially in unfamiliar areas. Tourism is an important aspect of the economy of any city and can be endangered by high crime rates.

Our scenario uses police department incident response data reported at `http://data.seattle.gov`, which logs each 911 call to the Seattle Police Department along with the geographic location, description of the incident, time of the incident, and various other pertinent information related to the call. We used the web service provided by the Seattle website as a Google Data Source to interface directly with MosaicJS using an RSS feed importer. In each entry in the feed, we only used the columns "time", "block location", "event description", "longitude", "latitude", and "event group." We also filtered out all records that involved minor incidents such as parking violations, noise violations, false alarms, and fraud. Figure 1 shows one example of a mosaic created based on this data. Note the regular grid pattern of incident locations; we speculate this is an artifact of a privacy-preserving mechanism where specific street locations are resolved to the closest grid location.

The most powerful aspect of this MosaicJS setup is that the feed is live; as the underlying Seattle data updates with new entries, the mosaic can simply be refreshed in order to be populated with new data. The Seattle Police Department data currently updates only every four hours, but other data feeds (such as fire response) updates every 10 minutes.

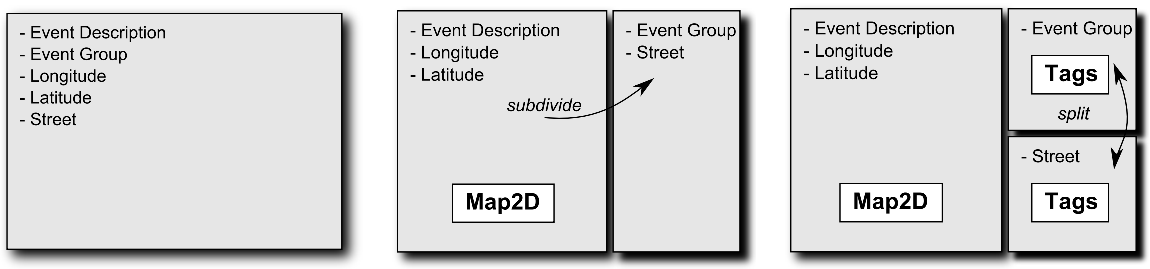
Tourists who are new to an area are often worried about getting lost in a bad neighborhood. Let us follow Tom, a tourist who is visiting Seattle for the first time. Worried about straying into the wrong neighborhood, Tom pulls up MosaicJS on his smartphone. He quickly loads the Seattle PD 911 incident dataset as a Google Data Source by cutting and pasting the URL into the data source field on the MosaicJS control panel (right side of Figure 3). This yields a single tile containing all of the dimensions in the data source. He can access the data management interface for this tile by selecting the "Data" tab for the tile at the upper right. This view lets Tom both split (create children) or subdivide (divide into two siblings) the tile based on the dimensions.

Using the data management interface, Tom splits the columns into a grouping that makes sense: he assigns "latitude", "longitude", and "event group" dimensions to a Google Map visualization that will plot all crime incidents as pushpins to a map of Seattle. This quickly gives him an idea of which parts are safe and which are not. Then he assigns the "event description" columns to a tag cloud, allowing him to quickly get an idea of the types of crimes that are most common in Seattle. He also plots "block location" as a tag cloud to see which streets the crimes are committed on. The process of setting up the mosaic is quick and painless. If Tom gets lost during his sightseeing stroll around Seattle, this tag cloud would help him know which street names to potentially avoid. In particular, if a riot would form during Tom's visit in the area, the visualization mosaic he just created (Figure 5) would provide him with dynamically updated data that allow him to find a safe way back to his hotel.

Let us consider Paul, a Seattle police officer, who could also use this application to quickly build visualization mosaics of dynamically updating data. He simply uses the built-in field laptop in his patrol car to connect to the MosaicJS website without having to install any new software. When Paul is on patrol, the mosaic serves as an "ear to the ground" for him to know what is currently going on in the city and the local neighborhood. When Paul gets a call, he can refresh the visualization and look up the location and its surrounding area to get a better situational awareness of the situation he is heading into. To avoid having to do this complex data management while responding to a call, Paul would presumably build the mosaic in beforehand (or have someone else build it for him), and only periodically refresh the mosaic to repopulate it with current data.

5.2. Rock Climbing

Rock and alpine climbing is becoming increasingly popular throughout the world, and many websites have been created such as `rockclimbing.com` (15,000 users), `14ers.com` (24,000 users), and `summitpost.org` (26,000 users). However, these websites currently have no easy way to allow their users to visually explore which mountains they would like to climb. Here we show how MosaicJS could be used to allow climbers to quickly narrow their searches for a



(a) Single tile with Seattle 911 police dataset. This was created by pasting the URL to the data feed into MosaicJS.

(b) Subdivide into geographical (left) and textual data (right) tiles. The right tile can now be assigned a Google map view, plotting events by name based on their position.

(c) Split textual data into separate tiles for events and streets. These tiles can now be assigned tag clouds as their visualization.

Figure 4: *Tile interaction operations used to go from the starting position (a single tile) to the result in Figure 5 through a subdivision followed by a split operation.*

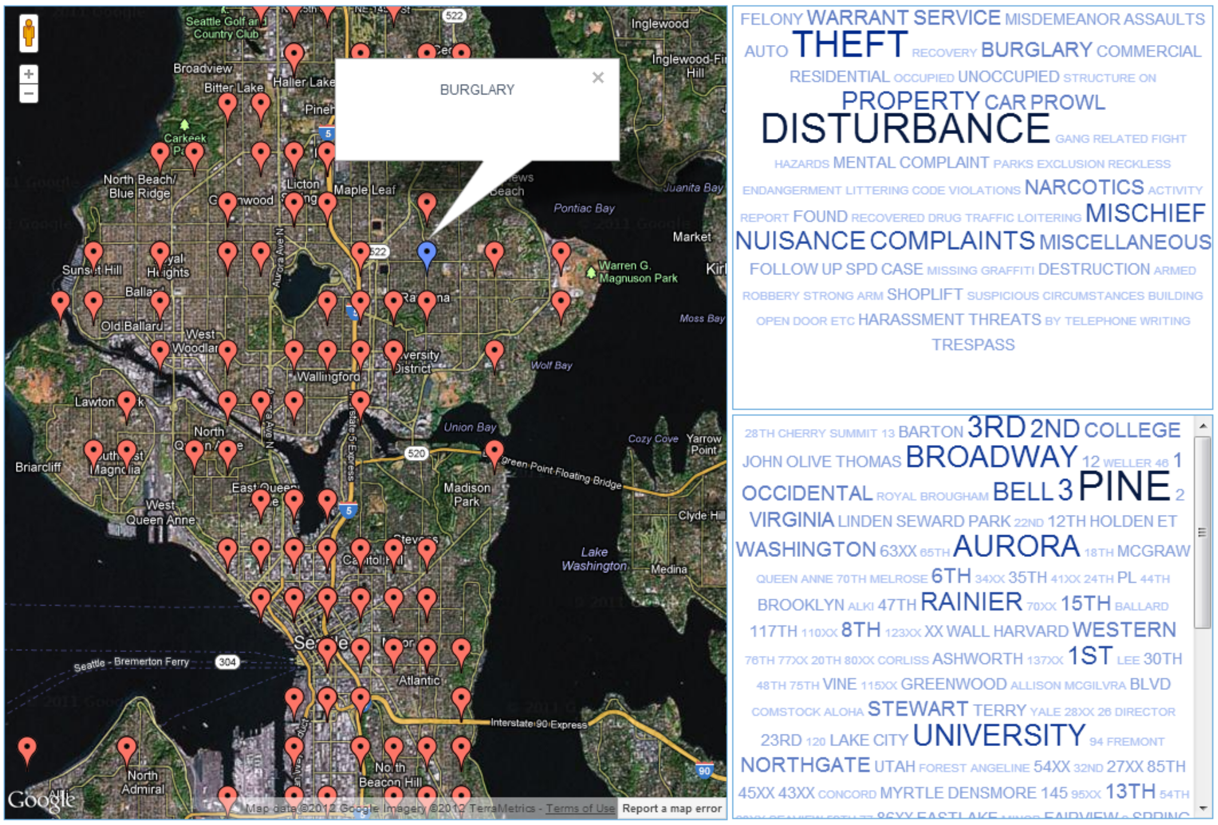
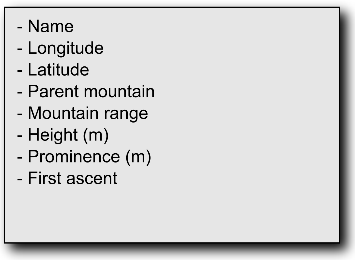

BURGLARY
FELONY WARRANT SERVICE MISDEMEANOR ASSAULTS
AUTO THEFT RECOVERY BURGLARY COMMERCIAL
RESIDENTIAL OCCUPIED UNOCCUPIED STRUCTURE ON
PROPERTY CAR PROWL
DISTURBANCE GANG RELATED FIGHT
HAZARDS MENTAL COMPLAINT PARKS EXCLUSION RECKLESS
ENDANGERMENT LITTERING CODE VIOLATIONS NARCOTICS ACTIVITY
REPORT FOUND RECOVERED DRUG TRAFFIC LOITERING MISCHIEF
NUISANCE COMPLAINTS MISCELLANEOUS
FOLLOW UP SPD CASE MISSING GRAFFITI DESTRUCTION ARMED
ROBBERY STRONG ARM SHOPLIFT SUSPICIOUS CIRCUMSTANCES BUILDING
OPEN DOOR ETC HARASSMENT THREATS BY TELEPHONE WRITING
TRESPASS
3RD 2ND COLLEGE
BROADWAY
OCCIDENTAL BELL PINE
VIRGINIA
WASHINGTON AURORA
RAINIER 15TH
8TH WESTERN
1ST
STEWART
UNIVERSITY
NORTHGATE

Figure 5: *Tom the tourist's mosaic of Seattle 911 police incidents. Figure 4 shows how this mosaic was created.*

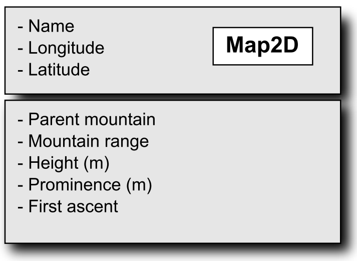
mountain that might fit their requirements, such as location, height, prominence, difficulty, and accessibility.

In our scenario, Chloe—a particularly tech-savvy climber—has adopted MosaicJS to explore the Himalayas, a large mountain range directly to the north of India. The Himalayas is the location of the world's tallest mountain, Mount Everest, along with other popular mountains such as K2. These mountains have seen increasing number of visitors after being featured in numerous movies lately.

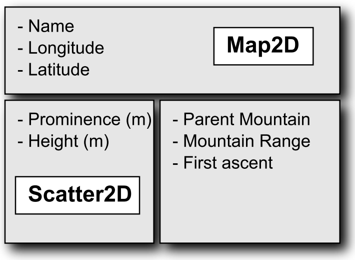
Chloe is already on location in Nepal and only has ac-



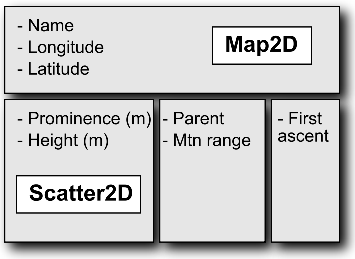
(a) Single tile with rock climbing data. This initial tile was created by pasting the datafeed URL into MosaicJS.



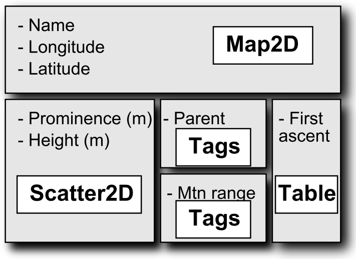
(b) Subdivide into geographical (top) and non-spatial data (bottom) tiles. The top tile can now be assigned a Google map view to plot peaks given their longitude and latitude and using their name as a label.



(c) Split the bottom tile, saving prominence and height in the left tile. This tile can now be assigned to a scatterplot, using index number of the peaks for the X axis and plotting height and prominence on the Y.



(d) Subdivide the bottom right tile to create a new sibling tile for first ascent data. This lower right tile can now be assigned a simple table that will show the year each peak (labeled by their index) was first ascended.



(e) Split the bottom center tile to create two subtiles, one for parent mountains and one for mountain ranges. Both of these tiles can now be assigned tag clouds to visualize the textual data.

Figure 6: *Tile interaction operations used to go from the starting position (a single tile) to the result in Figure 7.*

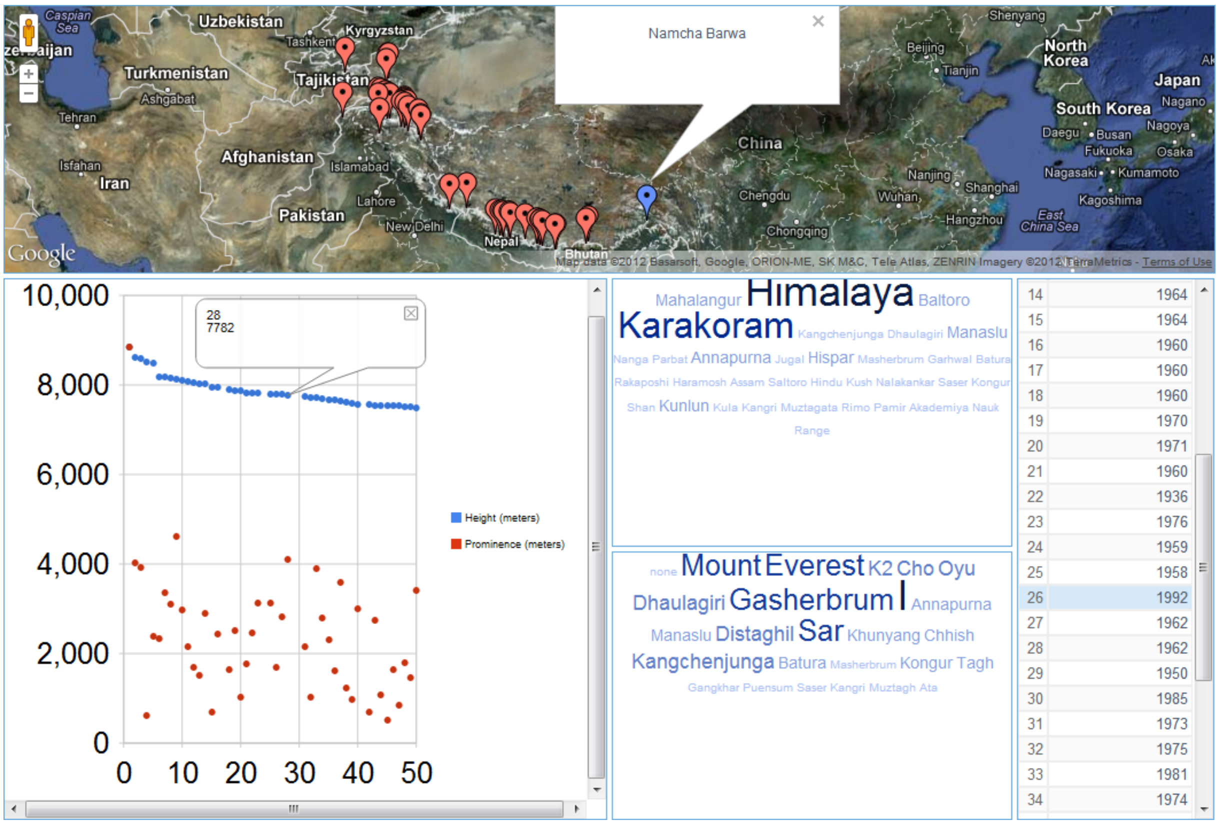
cess to her iPad, but still wants to study her climbing plans in detail. Fortunately, MosaicJS is based on pure HTML5, JavaScript, and SVG, and thus gives her access to advanced visualization capabilities even in the field and on a mobile device. She thus connects to a popular rock climbing website that already provides its users with a mountain dataset containing Mountain Rank (a number that corresponds to the mountains height in relation to the other mountains in this range), Name, Height (in meters), Mountain Range (the range which the mountain belongs to), Latitude, Longitude, Prominence (in meters, the elevation between summit and the lowest point uninterrupted by another summit), Parent Mountain, and First Ascent (the year in which the first person reached the summit).

Figure 7 shows how Chloe might use MosaicJS to explore this dataset. She first uses geomaps to plot the location of these peaks using the Mountain Name, Longitude, and Latitude. She then creates two tiles for the mountain range and parent mountain to get an idea of where the largest concentration of peaks exist. For smaller mountains, this allows her to plan climbing multiple mountains in one trip if they are all concentrated to one range. Chloe can also plot the dates of first ascent on a table to see how recently the range was first climbed, or if it remains unclimbed. Finally, she can plot the height and prominence by rank which allows her to see how high the summit is as well as how high she would have to climb to get to the summit. In a fictional social data analysis website built on MosaicJS, she could even share her findings to her fellow climbers, allowing them to comment and annotate the tentative plans and make a definite decision on the next step in their rock climbing adventure.

6. Discussion

The main strengths of our mosaics concept are the following:

- Allows for progressively building interactive web-based dashboards for heterogeneous datasets using simple and quick interactions;
- The structure and dataflow in the dashboard is visible from its hierarchical layout (shown using the space enclosure in the resulting mosaic); and
- Uses a common data model but places no restrictions on the visual components used in the individual tiles.


Namcha Barwa
28
7782
Height (meters)
Prominence (meters)
10,000
8,000
6,000
4,000
2,000
0
0 10 20 30 40 50
Mahalangur Himalaya Baltoro Karakoram Kangchenjunga Dhaulagiri Manaslu Nanga Parbat Annapurna Jugal Hispar Masherbrum Garhwal Batura Rakaposhi Haramosh Assam Saltoro Hindu Kush Nalakankar Saser Kongur Shan Kunlun Kula Kangri Muztagata Rimo Pamir Akademiya Nauk Range
none Mount Everest K2 Cho Oyu Dhaulagiri Gasherbrum I Annapurna Manaslu Distaghil Sar Khunyang Chhish Kangchenjunga Batura Masherbrum Kongur Tagh Gangkhar Puensum Saser Kangri Muztagh Ata

14	1964
15	1964
16	1960
17	1960
18	1960
19	1970
20	1971
21	1960
22	1936
23	1976
24	1959
25	1958
26	1992
27	1962
28	1962
29	1950
30	1985
31	1973
32	1975
33	1981
34	1974

Figure 7: *Rock climbing data dynamically visualized using MosaicJS. The user has selected one of the peaks—Namcha Barwa—in the Tibetan Himalaya. The other views (scatterplot on the lower left and the table on the lower right) have highlighted this peak, showing that it has a 7,782 meter elevation and was first ascended in 1992 (by a China-Japan expedition). Figure 6 illustrates the procedure to reach this point.*

In particular, one of the contributions of our mosaics concept is that it is an alternative to the classic coordinated multiple view (CMV) [Rob07] methodology, where the user manually creates (and deletes) views, assigns data dimensions, and arranges them in a 2D layout. The mosaic approach instead starts with a single view, or tile, that is progressively split into component tiles in a space-filling layout while automatically connecting the data dimensions during this refinement process. While we make no claim to the utility of this approach, it does minimize drag-and-drop, resizing, and layout interactions, which can be cumbersome and taxing for complex views. Furthermore, the state of a mosaic can be captured in a compact formal notation, which is suitable for encoding in a URL. This makes mosaics more suitable for browsers than classic CMV given the constraints of the web browser for supporting such interactions.

Of course, there are several limitations to our framework as well. For one thing, given our emphasis on web-based visualization, MosaicJS has limited scalability, certainly not to the level of frameworks such as VisBricks [LSS*11] or WebCharts [FDFR10]. Furthermore, even if we target a novice audience, the user is still expected to understand the MosaicJS data model, and also need to have an idea of which visual representation to choose for a particular tile.

Extensibility is another limitation. While it is true that our implementation can encapsulate virtually any existing web-based visualization component or toolkit, a dedicated JavaScript stub must be written to handle data conversion for each new component. In addition, even with this in place, it is far from certain that a given third-party component can be configured to understand mosaic-wide operations such as brushing, selecting, and highlighting. More work on visualization interoperability—particularly in the unifying ecosystem of web software and standards—is needed here.

Finally, visual complexity and clutter [ED07] is an issue with any composite visualization [JE12], and our visualization mosaics technique is no exception. In particular, while our slice-and-dice layout does convey the structure of the mosaic, it is also true that this structure can be difficult to perceive when the number of visualization tiles and the corresponding visual complexity increases [DGH03,Shn92]. This is the price to pay when combining multiple visual representations—the resulting composite visualization will naturally become more complex.

7. Conclusion and Future Work

We have proposed a new component-based approach to creating interactive information dashboards consisting of lightweight and easily modifiable visual representations that fit together as a *mosaic* of views instead of as a single, monolithic visualization application. Our implementation of visualization mosaics is called MosaicJS, and is an HTML/JavaScript framework that runs in a web browser and integrates several external components from the Google Visualization API, RaphaelJS, Protovis [BH09], and D3 [BOH11]. We showcased MosaicJS through examples involving 911 incident analysis and rock climbing.

The future outlook of web-based visualization is very promising as the transition to browser-based applications continues. However, the challenge with web-based visualization is to achieve the same interactivity, flexibility, and performance in the browser as in a standard application, while leveraging the social, modular, and community-driven features of the Internet. Our visualization mosaics are one step in the right direction, but more work is needed to fully be able to replace traditional visualization applications.

Acknowledgements

This work was partly funded by Google, Inc. under the research project "Visualization Mosaics: Effortless View Creation for Sensemaking."

References

[AEN10] ANDREWS C., ENDERT A., NORTH C.: Space to think: Large, high-resolution displays for sensemaking. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (2010), pp. 55–64.

[Ahl96] AHLBERG C.: Spotfire: An information exploration environment. *SIGMOD Record 25*, 4 (1996), 25–29.

[BC87] BECKER R. A., CLEVELAND W. S.: Brushing scatterplots. *Technometrics 29*, 2 (1987), 127–142.

[BCS96] BECKER R. A., CLEVELAND W. S., SHYU M.-J.: The visual design and control of trellis display. *Journal of Computational and Graphical Statistics 5*, 2 (1996), 123–155.

[BH09] BOSTOCK M., HEER J.: Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1121–1128.

[BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics 17*, 6 (2011). to appear.

[BWK00] BALDONADO M. Q. W., WOODRUFF A., KUCHINSKY A.: Guidelines for using multiple views in information visualization. In *Proceedings of the ACM Conference on Advanced Visual Interfaces* (2000), pp. 110–119.

[CM84] CLEVELAND W. S., MCGILL R.: Graphical perception: Theory, experimentation and application to the development of graphical methods. *Journal of the American Statistical Association 79*, 387 (Sept. 1984), 531–554.

[CMS99] CARD S. K., MACKINLAY J. D., SHNEIDERMAN B. (Eds.): *Readings in information visualization: Using vision to think*. Morgan Kaufmann Publishers, San Francisco, 1999.

[CWRY06] CUI Q., WARD M., RUNDENSTEINER E., YANG J.: Measuring data abstraction quality in multiresolution visualizations. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (Sept./Oct. 2006), 709–716.

[DCCW08] DÖRK M., CARPENDALE M. S. T., COLLINS C., WILLIAMSON C.: VisGets: Coordinated visualizations for web-based information exploration and discovery. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1205–1212.

[DGH03] DOLEISCH H., GASSER M., HAUSER H.: Interactive feature specification for focus+context visualization of complex simulation data. In *Proceedings of the Eurographics/IEEE VGTC Symposium on Visualization* (2003), pp. 239–248.

[DGWC10] DÖRK M., GRUEN D. M., WILLIAMSON C., CARPENDALE M. S. T.: A visual backchannel for large-scale events. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1129–1138.

[ED07] ELLIS G., DIX A. J.: A taxonomy of clutter reduction for information visualisation. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1216–1223.

[EDF08] ELMQVIST N., DRAGICEVIC P., FEKETE J.-D.: Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1141–1148.

[EMJ*11] ELMQVIST N., MOERE A. V., JETTER H.-C., CERNEA D., REITERER H., JANKUN-KELLY T.-J.: Fluid interaction for information visualization. *Information Visualization 10*, 4 (2011), 327–340.

[EST08] ELMQVIST N., STASKO J., TSIGAS P.: DataMeadow: A visual canvas for analysis of large-scale multivariate data. *Information Visualization 7* (2008), 18–33.

[FDFR10] FISHER D., DRUCKER S. M., FERNANDEZ R., RUBLE S.: Visualizations everywhere: A multiplatform infrastructure for linked visualizations. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1157–1163.

[Fek04] FEKETE J.-D.: The infovis toolkit. In *Proceedings of the IEEE Symposium on Information Visualization* (2004), pp. 167–174.

[Few06] FEW S.: *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly, 2006.

[Fit60] FITZPATRICK P. J.: Leading British statisticians of the Nineteenth Century. *Journal of the American Statistical Association 55*, 289 (Mar. 1960), 38–70.

[FP02] FEKETE J.-D., PLAISANT C.: Interactive information visualization of a million items. In *Proceedings of the IEEE Symposium on Information Visualization* (2002), pp. 117–124.

[Fri07] FRIENDLY M.: A brief history of data visualization. *Handbook of Computational Statistics: Data Visualization III* (2007).

[Gru01] GRUDIN J.: Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems* (2001), pp. 458–465.

[GZA06] GOTZ D., ZHOU M. X., AGGARWAL V.: Interactive visual synthesis of analytic knowledge. In *Proceedings of the IEEE Symposium on Visual Analytics Science & Technology* (2006), pp. 51–58.

[HA06] HEER J., AGRAWALA M.: Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 853–860.

[HLAJ11] HO Q., LUNDBLAD P., ASTRÖM T., JERN M.: A web-enabled visualization toolkit for geovisual analytics. In *Proceedings of SPIE-IS&T Electronic Imaging* (2011), vol. 7868 of *SPIE*.

[HVW07] HEER J., VIÉGAS F. B., WATTENBERG M.: Voyagers and voyeurs: supporting asynchronous collaborative information visualization. In *Proceedings of ACM CHI Conference on Human Factors in Computing Systems* (2007), pp. 1029–1038.

[Ins85] INSELBERG A.: The plane with parallel coordinates. *The Visual Computer 1*, 2 (1985), 69–91.

[JE10] JAVED W., ELMQVIST N.: Stack zooming for multi-focus interaction in time-series data visualization. In *Proceedings of the IEEE Pacific Symposium on Visualization* (2010), pp. 33–40.

[JE12] JAVED W., ELMQVIST N.: Exploring the design space of composite visualization. In *Proceedings of the IEEE Pacific Symposium on Visualization* (2012), pp. 1–8.

[JJKM08] JUSUFI I., JUNUZI L., KERREN A., MILRAD M.: Visualization of content and semantical relations of geonotes. In *Proceedings of the IASTED International Conference on Visualization, Imaging, and Image Processing* (2008), pp. 131–136.

[KAF*08] KEIM D., ANDRIENKO G., FEKETE J.-D., GÖRG C., KOHLHAMMER J., MELANÇON G.: Visual analytics: Definition, process, and challenges. In *Information Visualization – Human-Centered Issues and Perspectives* (2008), Kerren A., Stasko J. T., Fekete J.-D., North C., (Eds.), vol. 4950 of *LNCS*, Springer, pp. 154–175.

[Kei00] KEIM D. A.: Designing pixel-oriented visualization techniques: Theory and applications. *IEEE Transactions on Visualization and Computer Graphics 6* (2000), 59–78.

[Kei02] KEIM D. A.: Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics 8*, 1 (2002), 1–8.

[KK94] KEIM D. A., KRIEGEL H.-P.: VisDB: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications 14*, 5 (Sept. 1994), 40–49.

[LSS*11] LEX A., SCHULZ H.-J., STREIT M., PARTL C., SCHMALSTIEG D.: VisBricks: Multiform visualization of large, inhomogeneous data. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (2011), 2291–2300.

[LWW90] LEBLANC J., WARD M. O., WITTELS N.: Exploring N-dimensional databases. In *Proceedings of the IEEE Conference on Visualization* (1990), pp. 230–237.

[Mac86] MACKINLAY J.: Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics 5*, 2 (Apr. 1986), 110–141.

[McK09] MCKEON M.: Harnessing the information ecosystem with wiki-based visualization dashboards. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1081–1088.

[MHS07] MACKINLAY J. D., HANRAHAN P., STOLTE C.: Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1137–1144.

[NCIS02] NORTH C., CONKLIN N., INDUKURI K., SAINI V.: Visualization schemas and a web-based architecture for custom multiple-view visualization of multiple-table databases. *Information Visualization 1*, 3-4 (2002), 211–228.

[NS00] NORTH C., SHNEIDERMAN B.: Snap-together visualization: A user interface for coodinating visualizations via relational schemata. In *Proceedings of the ACM Conference on Advanced Visual Interfaces* (2000), pp. 128–135.

[Rob07] ROBERTS J. C.: State of the art: Coordinated & multiple views in exploratory visualization. In *Proceedings of the International Conference on Coordinated Multiple Views in Exploratory Visualization* (2007).

[SDW09] SLINGSBY A., DYKES J., WOOD J.: Configuring hierarchical layouts to address research questions. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 977–984.

[SGL08] STASKO J. T., GÖRG C., LIU Z.: Jigsaw: supporting investigative analysis through interactive visualization. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology* (2008), pp. 131–138.

[Shn92] SHNEIDERMAN B.: Tree visualization with tree-maps: A 2-D space-filling approach. *ACM Transactions on Graphics 11*, 1 (Jan. 1992), 92–99.

[Shn96] SHNEIDERMAN B.: The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Symposium on Visual Languages* (1996), pp. 336–343.

[SLBC03] SWAYNE D. F., LANG D. T., BUJA A., COOK D.: GGobi: evolving from XGobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis 43*, 4 (2003), 423–444.

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

[TFF88] TUKEY J. W., FISHERKELLER M. A., FRIEDMAN J. H.: PRIM-9: An interactive multi-dimensional data display and analysis system. In *Dynamic Graphics for Statistics*, Cleveland W. S., McGill M. E., (Eds.). Wadsworth & Brooks/Cole, 1988, pp. 111–120.

[TU08] THEUS M., URBANEK S.: *Interactive Graphics for Data Analysis: Principles and Examples (Computer Science and Data Analysis)*. Chapman & Hall/CRC, 2008.

[Tuk77] TUKEY J. W.: *Exploratory data analysis*. Addison-Wesley, 1977.

[VWvH*07] VIÉGAS F. B., WATTENBERG M., VAN HAM F., KRISS J., MCKEON M. M.: ManyEyes: a site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1121–1128.

[War94] WARD M. O.: XmdvTool: Integrating multiple methods for visualizing multivariate data. In *Proceedings of the IEEE Conference on Visualization* (1994), pp. 326–333.

[Wat05] WATTENBERG M.: Baby names visualization, and social data analysis. In *Proceedings of the IEEE Symposium on Information Visualization* (2005), pp. 1–6.

[Wea04] WEAVER C.: Building highly-coordinated visualizations in Improvise. In *Proceedings of the IEEE Symposium on Information Visualization* (2004), pp. 159–166.

[YMSJ05] YI J. S., MELTON R., STASKO J., JACKO J.: Dust & Magnet: Multivariate information visualization using a magnet metaphor. *Information Visualization 4*, 4 (2005), 239–256.

[YRW07] YANG D., RUNDENSTEINER E. A., WARD M. O.: Analysis guided visual exploration to multivariate data. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology* (2007), pp. 83–90.