Doctoral Dissertation

Curran Kelleher

4/23/2014

Abstract

There is immense potential value in data that is not being realized. While many data sets are available, it is difficult to realize their full value because they are made available using many different formats, protocols and vocabularies. The heterogeneity of formats, protocols and vocabularies makes it difficult to combine data sets together and hinders the development of data visualization software. While it is straightforward to produce static visualizations of just about any data set by customizing existing examples, there is a lack of generalized visualization software that supports the creation of interactive visualizations and visualization dashboards with multiple linked views. The contribution of this dissertation is a collection of data structures and algorithms supporting integration and interactive visualization of many data sets using interactive visualization dashboards with multiple linked views. A proof of concept implementation demonstrates support for several public data sets and well known visualization techniques.

Contents

Chapter 1

Introduction

The contributions of this dissertation are novel data structures and algorithms for integration and interactive visualization of many data sets from multiple sources, based on the data cube concept. The proposed data representation framework will allow data sets to be combined together and visualized using interactive visualization dashboards like the one described above, giving users the sense that the data exists within a single unified structure. The framework is designed to be able to represent and integrate an arbitrary number of data sets created independently of one another, and expose the integrated structure to reusable visualization tools that can be combined together in dashboard layouts with multiple linked views using existing interaction techniques such as brushing and linking. The proposed data representation and visualization framework is fundamentally new, and will allow heterogeneous data sets to be explored in a unified way that was never before possible.

The overall goal of this work is to build digital telescope into the universe of phenomena on Earth via publicly available data. For example, consider public data sources such as the United Nations, the US Census, the US Bureau of Labor Statistics, or the US Centers for Disease Control. These organizations and hundreds of others around the world provide publicly available data about various topics including population statistics, public health, distribution of wealth, quality of life, economics, the environment, and many others. By unifying these data sources and providing users with tools to explore and present the data visually, a deeper understanding of the world can be gleaned through the lens of public data. The focus of this dissertation is on applications involving public data, however the techniques introduced can be applied to any data sets that can be conceptually modeled as data cubes, regardless of whether they are public or private.

Consider the data from the US Census that covers population statistics for US States from 1950 to 2010. Consider also population statistics from the United Nations covering World Countries from 1970 to 2012. These two data sets may use different identifiers for years and geographic regions, but they cover an overlapping conceptual data space of time, geography and population. From these two data sets it is possible to create a visualization dashboard with a map of the world showing population as color and a corresponding line graph showing population for each region as lines. If the user views the whole world, the UN population data is shown for each country.

If the user zooms into the US, US Census data is shown for each state. If the user selects a point of time in the line graph, the data shown on the map is from that point in time. If the user pans and zooms on the map, the lines in the line graph update to only show the regions visible on the map. This is one example of an interactive visualization dashboard with multiple linked views (the timeline and map views) operating over multiple data sets integrated from different sources (the United Nations population data and the US Census population data).

TODO implement this and put screenshot image here

Data cubes, also known as OLAP (OnLine Analytical Processing) cubes, can represent data that contains measures aggregated (typically using sum or average) along categorical hierarchies. The data cube concept emerged from the field of data warehousing as a way to summarize transactional data, allowing analysts to get a bird's eye view of company activities. The term OLAP stands in contrast to the term OLTP (OnLine Transaction Processing), which is the part of the data warehouse system that ingests and stores data at the level of individual transactions or events. After the ETL (Extract, Transform and Load) phase of the data warehouse flow, the data is analyzed by computing a data cube from the transactional data.

The data cube concept and structure can be used to model existing data sets as well. Publicly available data sets (often termed "statistical data") may be considered as pre-computed data cubes if they contain aggregated measures (also called "indicators", "metrics" or "statistics") across time, geographic space or other dimensions such as gender, age range, ethnicity or industry sector. Any categorization scheme containing distinct entities, organized as an unorered collection, an ordered collection, or a hierarchy can be modeled as a dimension. Any numeric value that represents an aggregated statistical summary using sum, average, or other aggregation operator can be modeled as a measure.

With this approach, it is possible to model many data sets together using shared dimensions and measures. This will allow integration of many data sets together in a single unified structure. Existing data cube technologies assume that data cubes will be computed from a relational source, and are not designed to handle integration of pre-computed data cubes that may use inconsistent identifiers for common dimensions and inconsistent scaling factors for common measures. Therefore the application of the data cube concept to integration and visualization of many pre-computed data cubes, while theoretically plausible, requires the development of novel data structures and algorithms that extend the data cube model to handle integration of pre-computed data cubes that may use inconsistent identifiers for common dimensions and inconsistent scaling factors for common measures.

The data cube structure lends itself particularly well to visualization. Long standing perception-based data visualization theory presented by Bertin [3] and Mackinlay [20] identify effective ways to visually encode data based on data fields that are nominal, ordinal or quantitative. Visualization techniques have been explored for hierarchical (tree-based) data as well [11]. Data cubes can contain data of these types. Therefore existing visualization theory can be applied to the data cube model to determine which visualizations are appropriate for representing which data, depending on its data cube structure. This topic is discussed in section 8.3.

1.1 Related Work

Previous work relating to this dissertation falls into four major categories:

- data representation - structures, models and formats for data
- data integration - merging data from many sources
- data visualization - transforming data into interactive graphics
- Web graphics technology - HTML5 graphics APIs and libraries

1.1.1 Data Representation

Relational database systems provide a mature data management solution and are widely adopted [21]. The relational model has well understood theoretical underpinnings such as relational algebra [6]. Data warehouse systems are typically built on the relational model and are augmented by multi-scale aggregated data structures called data cubes, also known as OLAP (OnLine Analytical Processing) cubes [12, 7]. Data cubes contain summaries of the collection of facts stored in a relational database [5]. For example, a data cube may contain how much profit was made from month to month subdivided by product category, while the relational database may contain the information associated with each individual transaction.

Because data cubes provide an abstraction that handles aggregation, they are a widely used method of data abstraction for supporting visualization and analysis tasks [23]. Kimball pioneered the area of "Dimensional Modeling", which concerns constructing data warehouse schemas amenable to data cube construction and analysis [16]. Data cubes have been implemented in a variety of different systems, so effort has been made to discover unified conceptual or mathematical models that can characterize many implementations [9, 25, 24, 19, 2, 13, 4]. The data cube structure has also been used to model user Web browsing sessions to support data mining algorithms for Web prefetching [27].

1.1.2 Data Integration

1.1.3 Data Visualization

1.1.4 Web Graphics Technology

TODO pull all related work from proposal

1.2 Pseudocode Conventions

Throughout this document, pseudocode is used to express data structures and algorithms. Our pseudocode is similar to that found in the book "Introduction to Algorithms" [8], but differs significantly in that it uses a functional style. Primitive types in our pseudocode include numbers, strings, booleans, arrays, objects and functions. The following examples demonstrate the features of our pseudocode language.

```
1  $x = 5$
```

Line numbers appear to the left of each line of pseudocode. Variables can have any name comprised of characters without spaces, and can be assigned a value with the = symbol. Variables need not be explicitly declared. The scope of a variable is determined by where it is first assigned. Our pseudocode uses block scope, meaning that every indentation level introduces a new nested scope. On line 1 of the above pseudocode example, the variable x is defined and assigned the value of 5, a numeric literal.

The following pseudocode demonstrates numbers, strings and booleans.

```
1  $myNumber = 5$
2  $myString =$ 'test'
3  $myBoolean =$ TRUE
4  $myOtherBoolean =$ FALSE
```

All numbers are treated as double precision floating point. Numeric literals in pseucode become numbers (see line 1). String literals are denoted by single quotes and a monospace font (see line 2). Booleans can be either true or false. True and false are builtin constant boolean values denoted by all capitalized words (see lines 3 and 4). Camel case names starting with a lower case letter are used for most variables in our pseudocode.

```
1 $add = \lambda(a, b)$
2         **return** $a + b$
3 $result = add(4, 6)$ // $result$ is assigned the value 10
4 $triple = \lambda(x)$ **return** $x * 3$
5 $triple(3)$ // evaluates to 9
```

The above pseudocode demonstrates how a function is defined and invoked, and also introduces comments. This example defines a function called *add* that adds two numbers together. The λ symbol defines a new anonymus function. Variables can be assigned functions as values using =. The comma separated names in parentheses directly following the λ are the arguments to the function. The pseudocode on lines following the λ that is indented one level constitutes the function body (also called the function closure). The function arguments are only visible inside the function closure.

Functions can be invoked using parentheses. The argument values are passed to the function in a comma separated list within parentheses. On line 3, the *add* function is invoked, passing the value 4 as argument a and 6 as argument b. The value returned by the function is assigned to the variable $result$. The function invocation causes the function body to execute, which adds the two numbers together and returns the resulting number using the "return" keyword on line 2. Lines 4 and 5 demonstrate that a simple anonymous function can be defined in a single line. Text following the // symbol is a comment, and is not executed.

```
1  $myArray = []$
2  $myArray.push(5)$ // $myArray$ now contains $[5]$
3  $myArray.push(7)$ // $myArray$ now contains $[5, 7]$
4  $myArray.push(9)$ // $myArray$ now contains $[5, 7, 9]$
5  $myArray[0]$ // evaluates to 5
6  $myArray[2]$ // evaluates to 9
7  $myArray[1] = 3$ // $myArray$ now contains $[5, 3, 9]$
8  $myBooleanArray = [\text{TRUE}, \text{FALSE}, \text{TRUE}, \text{TRUE}]$
9  $myStringArray =$ ['foo','bar']
10 $numberOfBooleans = myBooleanArray.length$ // evaluates to 4
11 $numberOfStrings = myStringArray.length$ // evaluates to 2
12 for $str \in myStringArray$
13     $log(str)$ // also prints 'foo', then prints 'bar'
14 $myArray.map(triple)$ // evaluates to $[15, 21, 27]$
```

The above pseudocode demonstrates arrays. Arrays are ordered lists of elements. Arrays can contain elements of any type. Array literals are denoted by square brackets and can be empty (as in line 1) or populated (as in lines 8 and 9). Arrays have a built-in function attached to them called *push*, which appends a new element to the end of the array. Lines 2-4 demonstrate how *push* can be used to append items to an array. The dot notation seen on lines 2-4 is used on arrays only to access built-in functions and properties.

Square brackets denote access of array elements by index when placed directly after the variable name of the array. Array indices start at zero.

Lines 5 and 6 demonstrate how square bracket notation can be used to access values in an array based on their index. Line 7 demonstrates that square bracket notation can also be used to assign to values in an array. Lines 10 and 11 demonstrate the built-in property *length*, the number of elements in the array.

TODO explain lines 12 - 15

```
$myObject = \{\}$
$myObject.first =$ 'John' // $myObject$ now contains $\{first :$ 'John'$\}$
$myObject[$'last'$] =$ 'Doe' // now $\{first :$ 'John'$, last :$ 'Doe'$\}$
$myOtherObject = \{first :$ 'Jane'$, last :$ 'Doe'$\}$
$box =$
    $x : 50$
    $y : 60$
    $width : 100$
    $height : 150$
$properties = box.keys$ // evaluates to [x,y,width,height]
$values = properties.map(\lambda(property)$ **return** $box[property])$
// $values$ is assigned $[50, 60, 100, 150]$
```

TODO explain above code

```
$run(\lambda()\, log($'b'$))$
$log($'a'$)$
// Prints a, then b
```

Our pseudocode assumes a single threaded execution environment with a built-in event loop, which may be implemented using the reactor pattern [22]. The event loop can be used to queue functions to be executed in the future. In our pseudocode, the *run* built in function provides access to the event loop. Calling *run* and passing a function queues that function to be invoked in the future, after the current codepath terminates and all previously queued functions finish executing. In the above pseudocode, line

TODO discuss NIL **TODO** Discuss apply(fn, args) **TODO** Discuss !bool

Chapter 2

Functional Reactive Models

Functional reactive programming allows developers to declaratively specify data dependency graphs [26]. Functional reactive programming has been applied to interactive graphics [10] and robotics [15]. The Model View Controller paradigm is an approach to cleanly separate application operations into three classes. The *model* contains the data structures representing the application state. The *view* handles graphical presentation of the model to the user. The *controller* translates user interactions (such as mouse clicks, key presses, or multi-touch gestures) into changes in the model [17]. The Model View Controller paradigm can be combined with functional reactive programming to enable straightworward creation of reactive systems based on data flow graphs.

2.1 The Model Data Structure

The Model in the Model View Controller (MVC) paradigm is responsible for:

- managing the state of the application,
- allowing the Controller to change the state of the application, and
- notifying the view when the state of the application changes.

One simple and widely used method for structuring a Model is as a set of key-value pairs [18]. This kind of model can fulfill the all of the responsibilities of a Model with three methods:

- $set(key, value)$ Set the value for a given key.
- $get(key)$ Get the value for a given key.
- $on(key, callback)$ Add a change listener for a given key. Here, *callback* is a function that will be invoked synchronously when the value for the given key is changed.

This approach is used by popular JavaScript frameworks such as Backbone and Knockout. Using our pseudocode conventions, we will discuss one way a key-value Model can be implemented. We will first discuss a simple version with only *set* and *get*, then discuss a more complex version that also includes *on*.

```
1 $SimplestModel = \lambda()$
2     $values = \{\,\}$
3     **return**
4         $set : \lambda(key, value)\, values[key] = value$
5         $get : \lambda(key)$ **return** $values[key]$
```

The above pseudocode implements a key-value model that has only *set* and *get* methods. Line 1 defines the constructor function, *SimplestModel*, which will return a new object that has *set* and *get* methods. Line 2 defines a private variable called *values* that will contain the key-value mapping. Lines 3 - 5 define the *set* and *get* methods, which store and retreive values from the internal *values* object. Here's an example of how *SimplestModel* might be used.

```
$mySimplestModel = SimplestModel()$
$mySimplestModel.set($'x'$, 5)$
$mySimplestModel.get($'x'$)$ // Evaluates to 5
```

Here is a version of the model that implements the *on* method as well:

```
1  $SimpleModel = \lambda()$
2      $values = \{\,\}$
3      $callbacks = \{\,\}$
4      **return**
5          $on : \lambda(key, callback)$
6              **if** $callbacks[key] == \textsc{nil}$
7                  $callbacks[key] = [\,]$
8              $callbacks[key].push(callback)$
9          $set : \lambda(key, value)$
10             $values[key] = value$
11             **if** $callbacks[key] \neq \textsc{nil}$
12                 **for** $callback \in callbacks[key]$
13                     $callback()$
14         $get : \lambda(key)$ **return** $values[key]$
```

The above version includes an additional private variable, *callbacks*, which is an object whose keys are property names and whose values are arrays of callback functions. The *on* method defined starting at line 5 adds the given callback to the list of callbacks for the given key (and creates the list if it does not yet exist). The *set* method has been modified to invoke the callback functions associated with the given key when the value for that key is changed. Here is an example of how the *on* method can be used.

```
1 mySimpleModel = SimpleModel()
2 mySimpleModel.on('x', λ()
3         log(mySimpleModel.get('x'))
4 )
5 mySimpleModel.set('x', 5) // Causes line 3 to log 5
6 mySimpleModel.set('x', 6) // Causes line 3 to log 6
```

2.2 Functional Reactive Change Propagation

For complex applications such as interactive visualizations, managing propagation of changes can quickly become complex. For this reason, modular visualization environments based on data flow have become popular [1]. A data flow graph defines a directed acyclic graph of data dependencies. The data flow model is amenable to construction of visual programming languages [14]. While many systems consider data flow as a means to construct data transformation pipelines, the concept also applies to building reactive systems that manage change propagation throughout an application or subsystem in response to user interactions or other events [10].

To provide a solid foundation for dynamic visualization systems, the Model should be able function in the context of data dependency graphs. Developers should be able to declaratively specify data dependencies, and change propagation should be automatically managed. The *when* operator from functional reactive programming propagates changes from one or more reactive functions (such as is found in the JavaScript libraries Bacon.js and

RXJS).

Our Model implementation can be extended with a *when* operator that enables construction of data dependency graphs. This operator will become a foundation for building dynamic interactive visualizations. Since *when* is superior to *on* in that it handles change propagation intelligently, in this final version *on* is not exposed in the public Model API.

```
$Model = \lambda()$
    $simpleModel = SimpleModel()$
    **return**
        $set : simpleModel.set$
        $get : simpleModel.get$
        $when : \lambda(dependencies, fn)$
            $callFn = debounce(\lambda()$
                $args = dependencies.map(simpleModel.get)$
                **if** $allAreDefined(args)$
                    $apply(fn, args)$
            $)$
            $callFn()$
            **for** $key \in dependencies$
                $simpleModel.on(key, callFn)$
```

TODO describe Model()

```
$debounce = \lambda(fn)$
    $queued = $ FALSE
    **return** $\lambda()$
        **if** $queued ==$ FALSE
            $queued = $ TRUE
            $run(\lambda()$
                $queued = $ FALSE
                $fn()$
            $)$
```

TODO describe debounce()

```
$allAreDefined = \lambda(arr)$
    **for** $item \in arr$
        **if** $item$ is undefined
            **return** FALSE
    **return** TRUE
```

TODO describe allAreDefined()

TODO add pseudocode for each figure

2.3 Functional Reactive Visualizations

Figure 2.1: A simple data dependency graph using functional reactive models. Here, $fullName$ is recomputed whenever $firstName$ or $lastName$ change.

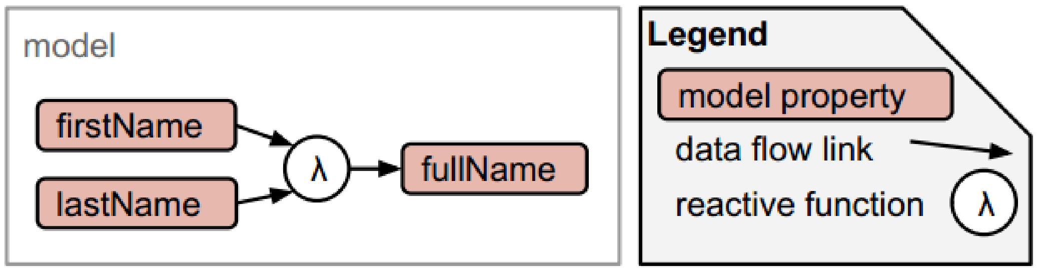


Figure 2.2: A data dependency graph with two hops. When x changes, the change propagates to y then to z.

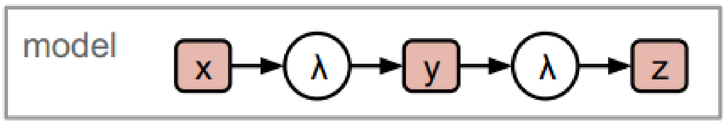
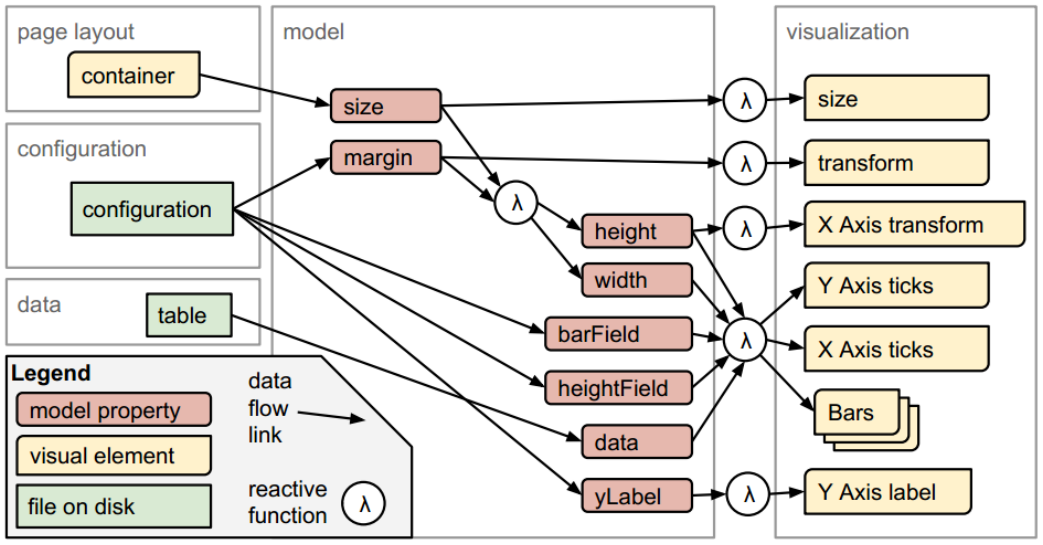


Figure 2.3: The data dependency graph for a dynamic bar chart.



Chapter 3

The Universal Data Cube

Chapter 4

Data Sets

Chapter 5

Visualizations

Chapter 6

Visualization Dashboard Infrastructure

Chapter 7

UDC Visualization Dashboards

Chapter 8

Collaboration and History Navigation

8.1 Related Work

Previous work relating to this dissertation falls into four major categories:

- data representation - structures, models and formats for data
- data integration - merging data from many sources
- data visualization - transforming data into interactive graphics
- Web graphics technology - HTML5 graphics APIs and libraries

8.1.1 Data Representation

Relational database systems provide a mature data management solution and are widely adopted [21]. The relational model has well understood theoretical underpinnings such as relational algebra [6]. Data warehouse systems are typically built on the relational model and are augmented by multi-scale aggregated data structures called data cubes, also known as OLAP (OnLine Analytical Processing) cubes [12, 7]. Data cubes contain summaries of the collection of facts stored in a relational database [5]. For example, a data cube may contain how much profit was made from month to month subdivided by product category, while the relational database may contain the information associated with each individual transaction.

Because data cubes provide an abstraction that handles aggregation, they are a widely used method of data abstraction for supporting visualization and analysis tasks [23]. Kimball pioneered the area of "Dimensional Modeling", which concerns constructing data warehouse schemas amenable to data cube construction and analysis [16]. Data cubes have been implemented in a variety of different systems, so effort has been made to discover unified conceptual or mathematical models that can characterize many implementations [9, 25, 24, 19, 2, 13, 4]. The data cube structure has also been used to model user Web browsing sessions to support data mining algorithms for Web prefetching [27].

8.1.2 Data Integration

8.1.3 Data Visualization

8.1.4 Web Graphics Technology

TODO pull all related work from proposal

8.2 Data Cube Representation and Integration

8.2.1 Defining Data Cubes

TODO describe CSV + JSON representation

8.2.2 Building a Data Cube Index

TODO pseudocode for building the index

8.2.3 Querying a Data Cube

TODO pseudocode for querying the index

8.2.4 Integration of Dimensions

TODO include modified pseudocode for UDC index building & query that resolves identifiers

8.3 Data Cube Visualization

8.3.1 Relating Data Cubes and Visualization Theory

Data cubes can contain the following kinds of data:

- *nominal* - unordered collections of categories
- *ordinal* - ordered collections of categories
- *hierarchical* - collections of categories organized as trees
- *quantitative* - continuously varying numeric values

Data cube dimensions can be nominal, ordinal or hierarchical. Data cube measures are always quantitative. This mapping relates data cubes to data types that have been well studied in the literature on visualization theory [3, 20, 11].

TODO add table with (nominal, ... , quantitative) vs. (color, value, position, size, connection, containment, ...)

8.3.2 A Visualization Taxonomy by Data Cube Structure

TODO add table with (Visualization, Data Cube Structure) referencing visualizations in other sections

8.4 Visualization Dashboard Infrastructure

8.4.1 Model Driven Visualizations

TODO include model driven bar chart pseudocode

8.4.2 Functional Reactive Visualizations

TODO include data flow graph for bar chart **TODO** Discuss the "when" Functional Reactive Operator **TODO** Cite functional reactive animation paper **TODO** include bar chart pseudocode using "when"

8.4.3 Dashboard Layout using Nested Boxes

TODO Include nested box layout pseudocode

8.4.4 Multiple Linked Views

TODO Include generic linking pseudocode using "when"

8.4.5 Dynamic Dashboard Configuration

TODO Include pseudocode for computing configuration diffs

8.5 Data Sets

8.5.1 United Nations Population Estimates

8.5.2 United Nations Millenium Development Goals

8.5.3 United States Census Population Estimates

TODO include figures/usCensusPopulationByState.png **TODO** import data from http://www.census.gov/popest/data/state/totals/2013/index.html

8.5.4 US Central Intelligence Agency World Factbook

TODO import subsets of the data

8.5.5 US Centers for Disease Control Causes of Death

TODO generalize stacked area and tree vis

8.5.6 W3Schools Browser Market Share

TODO import this data completely

8.5.7 Natural Earth

TODO include figures/naturalEarth.png **TODO** discuss data transformation process

8.6 Visualizations

TODO discuss pseudocode conventions **TODO** include pseudocode for each visualization

8.6.1 Bar Chart

8.6.2 Scatter Plot

8.6.3 Line Chart

8.6.4 Choropleth Map

8.6.5 Stacked Area Plot

8.6.6 TreeMap

8.6.7 Node Link Tree

8.6.8 Radial Tree

8.6.9 Icicle Plot

8.6.10 TreeMap

8.6.11 Parallel Coordinates

8.7 Visualization Dashboards

Bibliography

[1] Greg Abram and Lloyd Treinish. An extended data-flow architecture for data analysis and visualization. In *Proceedings of the 6th conference on Visualization'95*, page 263. IEEE Computer Society, 1995.

[2] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. Modeling multidimensional databases. In *Data Engineering, 1997. Proceedings. 13th International Conference on*, pages 232–243. IEEE, 1997.

[3] Jacques Bertin. Semiology of graphics: diagrams, networks, maps. 1983.

[4] Markus Blaschka, Carsten Sapia, Gabriele Hofling, and Barbara Dinter. Finding your way through multidimensional data models. In *Database and Expert Systems Applications, 1998. Proceedings. Ninth International Workshop on*, pages 198–203. IEEE, 1998.

[5] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[6] James Clifford and Abdullah Uz Tansel. On an algebra for historical relational databases: two views. In *ACM SIGMOD Record*, volume 14, pages 247–265. ACM, 1985.

[7] Edgar F Codd, Sharon B Codd, and Clynch T Salley. Providing olap (on-line analytical processing) to user-analysts: An it mandate. *Codd and Date*, 32, 1993.

[8] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*. MIT press, 2009.

[9] Anindya Datta and Helen Thomas. The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses. *Decision Support Systems*, 27(3):289–301, 1999.

[10] Conal Elliott and Paul Hudak. Functional reactive animation. In *ACM SIGPLAN Notices*, volume 32, pages 263–273. ACM, 1997.

[11] Martin Graham and Jessie Kennedy. A survey of multiple tree visualisation. *Information Visualization*, 9(4):235–252, 2010.

[12] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[13] Marc Gyssens and Laks VS Lakshmanan. A foundation for multi-dimensional databases. In *VLDB*, volume 97, pages 106–115, 1997.

[14] Daniel D Hils. Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing*, 3(1):69–101, 1992.

[15] Paul Hudak, Antony Courtney, Henrik Nilsson, and John Peterson. Arrows, robots, and functional reactive programming. In *Advanced Functional Programming*, pages 159–187. Springer, 2003.

[16] Ralph Kimball. *The data warehouse lifecycle toolkit: expert methods for designing, developing, and deploying data warehouses*. Wiley. com, 1998.

[17] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.

[18] Avraham Leff and James T Rayfield. Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, pages 118–127. IEEE, 2001.

[19] Chang Li and X Sean Wang. A data model for supporting on-line analytical processing. In *Proceedings of the fifth international conference on Information and knowledge management*, pages 81–88. ACM, 1996.

[20] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics (TOG)*, 5(2):110–141, 1986.

[21] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems.* Osborne/McGraw-Hill, 2000.

[22] Douglas C Schmidt. Reactor: An object behavioral pattern for concurrent event demultiplexing and dispatching. 1995.

[23] Chris Stolte, Diane Tang, and Pat Hanrahan. Multiscale visualization using data cubes. *Visualization and Computer Graphics, IEEE Transactions on*, 9(2):176–187, 2003.

[24] Panos Vassiliadis. Modeling multidimensional databases, cubes and cube operations. In *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on*, pages 53–62. IEEE, 1998.

[25] Panos Vassiliadis and Timos Sellis. A survey of logical models for olap databases. *ACM Sigmod Record*, 28(4):64–69, 1999.

[26] Zhanyong Wan and Paul Hudak. Functional reactive programming from first principles. In *ACM SIGPLAN Notices*, volume 35, pages 242–252. ACM, 2000.

[27] Qiang Yang, Joshua Zhexue Huang, and Michael Ng. A data cube model for prediction-based web prefetching. *Journal of Intelligent Information Systems*, 20(1):11–30, 2003.