The Case for Data Visualization Management Systems [Vision Paper]

Eugene Wu
sirrice@csail.mit.edu

Leilani Battle
leilani@csail.mit.edu

Samuel R. Madden
madden@csail.mit.edu

Most visualizations today are produced by retrieving data from a database and using a specialized visualization tool to render it. This decoupled approach results in significant duplication of functionality, such as aggregation and filters, and misses tremendous opportunities for cross-layer optimizations. In this paper, we present the case for an integrated Data Visualization Management System (DVMS) based on a declarative visualization language that fully compiles the end-to-end visualization pipeline into a set of relational algebra queries. Thus the DVMS can be both *expressive* via the visualization language, and *performant* by leveraging traditional and visualization-specific optimizations to scale interactive visualizations to massive datasets.

1. INTRODUCTION

The holy grail of visualization systems makes exploring different data facets so intuitive, and recommends views that are so relevant, that users rapidly converge onto valuable insights – irrespective of dataset size. Unfortunately, existing systems fall far short of this goal [16].

Most visualizations are produced by retrieving raw data from a database and using a specialized visualization tool to process and render it. Although the database can sometimes be used to filter the raw data (e.g., return data within a visible bounding box), visualization tools try to avoid roundtrips to the database by managing their own results cache and executing data transformations directly.

This decoupled approach has three drawbacks. First, the database is unaware of related queries and may recompute the same results. For example, slightly panning a map will issue a query to recompute the entire map, though most results are unchanged. Second, visualization tools duplicate basic database operations, such as filtering and aggregation. Some tools even implement an entirely new database for this purpose [28]. Lastly, visualization tools assume that all raw data and metadata fit entirely in memory (e.g., [1, 7, 20]). Thus their memory-based functionality cannot scale to massive datasets, which rarely fit in memory.

We propose instead to blend these two systems into a *Data Visualization Management System* (DVMS) to make all database features available for visualization. A DVMS scales common data transformations to massive datasets by executing them directly in the database. Furthermore, a DVMS can leverage databases to support interactivity with little effort. For example, with lineage query support, we can automatically link related geometric objects (e.g., circles, rectangles) across views by tracking overlap in the input records that generated them. DVMS's can also incorporate novel *visual optimizations* to reduce rendering latencies, such as: (1) applying occlusion filters to remove records that render as geometric objects hidden from the user's view; (2) output-based downsampling of datasets to match the viewport size; and (3) rendering on both the client and server to balance resource and network constraints.

We present the two central design ideas behind our proposed DVMS Ermac: (1) the user specifies a visualization workflow, or mapping from raw data to geometric objects, using a *declarative visualization language*; and (2) the workflow is compiled into relational algebra queries, which are executed by the database. Using this design, we can leverage traditional database optimizations to boost rendering and processing performance, and develop specialized optimizations based on semantic cues inferred from our workflows to scale interactive visual exploration to massive datasets.

2. THE Ermac SYSTEM

Ermac can be used as a standalone system, as a domain specific language within a general programming language such as Javascript or Python, or as the execution framework for specifications generated from visual direct manipulation tools such as Lyra [3]. The Ermac language borrows heavily from existing grammar-based languages [1, 30]. We now describe how a visualization specification is represented as a Logical Visualization Plan (LVP) that is compiled into a sequence of relational algebra queries that constitute a Physical Visualization Plan (PVP) [1]. The PVP is finally executed to produce a static visualization. Section 3.1 presents mechanisms to incorporate dynamic interactions.

Our example data is an `election` table containing Obama and Romney's campaign expenditures during the 2012 US presidential election. The table attributes include the candidate name, party affiliation, purchase dates within a 10

[1] The PVP will be further optimized and compiled by the DBMS into a traditional operator tree.

month period (Feb. to Nov. 2012), amount spent, and recipient:

election(*candidate*, *party*, *day*, *amount*, *recipient*)

2.1 Logical Visualization Plan

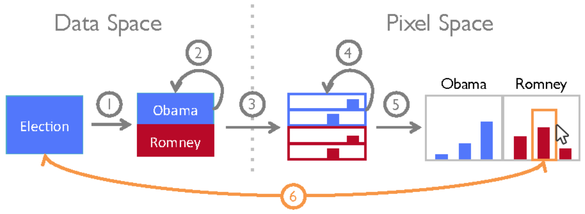

Figure 1: expenses Logical Visualization Plan.

Ermac executes a visual specification on a relational table (in data space) and generates a set of visual elements rendered as pixels on the screen (in pixel space). Figure 1 depicts the five stages of this process (grey arrows), where each stage has a corresponding visualization operator class. The operators are primarily distinguished by whether they occur in data space (arrows 1 and 2 split the table by candidate and summarize the data), pixel space (arrows 4 and 5 reposition the visual objects and render them), or between the two (arrow 3 maps data onto visual objects). This subsection briefly describes these five classes using an example visualization of the election table. The final orange arrow (6) represents visualization interaction, which we describe in Section 3.

Our syntax is a nested list of clauses, where each [class: operator] clause describes one of the five operator classes. Top level clauses define global operator bindings, and nested clauses are unique to a given layer (described below). Clauses may only be nested within layer, which cannot be nested within itself:

```
[class: operator]*      // top level clause
[layer:                 // layer clause
   [class: operator]*   // nested clause
]*
```

Listing 1: Specification to visualize election table

```
 1 data: election
 2 aesmap: x=day,y=amount
 3 layer:
 4   stat: [sort(on=x),cum]
 5   geom: line
 6 layer:
 7   stat: bin(bins=10)
 8   geom: rect
 9   //stat: bin(bins=DUMMY)
10 facet:
11   fx: candidate
12   fy: [DUMMY = (10,20)]
```

Lines 1-5 of Listing 1 are sufficient to render a line chart that shows total cumulative spending over time during the 2012 US presidential election. The data clause specifies the input table (which may also be a SQL SELECT query), the aesmap clause specifies the aesthetic mapping from the *day* and *amount* attributes to the x and y positional encodings, and the layer [2] clause describes how the mapped attributes should be rendered. For instance, stat clauses are data-space transformations; Line 4 first sorts the data by x (*day*) then computes the cumulative sum over y (amount) for each day. The geom clause in Line 5 renders the result as a line.

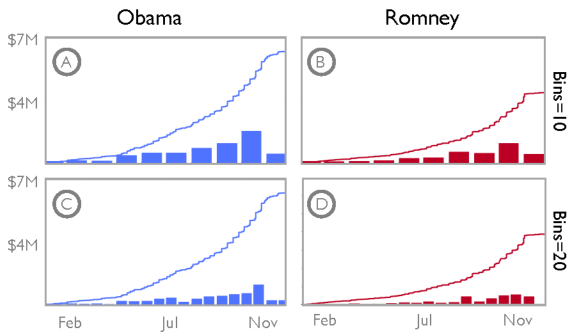

Figure 2: Faceted visualization of expenses table

Lines 6-8 render a new layer that contains a histogram of the total expenditures partitioned by day into ten buckets. The bin operator partitions the x attribute into ten equi-width bins (i.e., months) and sums the y values (Figure 2.A).

It makes sense to compare the purchasing habits of the two candidates side-by-side (Figures 2.A,B). The facet clause (Lines 10-11) specifies that the data is partitioned by *candidate* name; the visualization draws a separate *view*, or subfigure, for each partition; and the views are rendered as a single row along the x (fx) dimension.

It is often useful to compare visualizations generated from different operators or operator parameters (e.g., compare different sampling and aggregation techniques). Ermac's novel *parameter-based faceting* uses special DUMMY operators and parameters that are replaced at compile time. For example, Line 12 further divides the the visualization into a 2-by-2 grid (Figure 2.A-D), where each row varies the DUMMY operator in the specification. Thus, replacing Line 7 with 9 changes the bins parameter into a dummy variable that will be replaced with a binning value of either 10 (monthly) or 20 (bi-weekly), as dictated by Line 12.

2.2 Physical Visualization Plan

In this subsection, we describe the LVP's data and execution model and how an example logical operator (facet) is compiled into SQL queries that are part of the PVP.

Ermac's data model is nearly identical to the relational model, however we support data types that are references to rendered visual elements (e.g., SVG element). Thus, the data model can encapsulate the full transformation of input data records to records of visual elements that the user sees[3]. For example, to produce the above histogram, Ermac first aggregates the expenses into 10 bins, maps each bin (month) to an abstract rectangle record, and finally transforms the rectangle records to physical rectangle objects drawn on the screen. When the user specifies faceting or multiple layers, Ermac also augments the data relation with attributes (e.g., fx, fy, layerid) to track the view and layer where each record should be rendered.

Ermac additionally manages a scales relation that tracks the mapping from the domains of data attributes (e.g., *day*, *amount*) to the ranges of their corresponding perceptual encodings (e.g., x, y pixel coordinates). For instance, our example visualization linearly maps the *day* attribute's domain ($[Feb, Nov]$) to pixel coordinates ($[0, 100]$) along the x

[2] layers are listed in drawing order

[3] The physical rendering is modeled as UDFs that make OpenGL/WebGL calls

axis. These records are maintained for each aesthetic variable in every facet and layer.

Representing all visualization state as relational tables lets Ermac compile each logical operator into one or more relational algebra queries that take the data relation and scales relation as input and update one of the two relations. For example, Ermac reads the data relation to update the attribute domains in the scales relation, whereas data-space transformations (e.g., bin) read the x (*day*) attribute's domain from the scales relation to compute bin sizes.

Due to lack of space, we only describe how the facet operator is compiled and how it modifies the downstream LVP to deal with dummy variables. The fx: candidate clause (Line 11) partitions the data by *candidate* name and creates a unique facet attribute value for each partition. This is represented as a projection that creates a new data relation:

```
data = SELECT *, candidate as fx from data
```

The parameter-based faceting (Line 12) is compiled into a cross product with a custom table, facety(fy), that contains a record for each parameter value (e.g., 10 and 20):

```
SELECT data.*,facety.fy FROM data OUTER JOIN facety
```

Furthermore, facet replicates the downstream LVP for each fy value 10 and 20. If the fx clause were also a parameter list of size M, the downstream plan would be replicated $2M$ times – once for each pair of fx, fy values.

Although we have developed compilation strategies for all major logical operators, many of the relational queries rely on expensive cross-products or nested sub-queries. Many of these operations are unavoidable, regardless of whether Ermac or another system is creating the visualization. However, by expressing these expensive operations declaratively, we can use existing optimization techniques and develop new visualization techniques to improve performance. For instance, Ermac knows that queries downstream from parameter-based faceting will not update the data relation so it can avoid redundant materialization when executing the cross-product. Identifying further optimizations for individual and across multiple LVP operators poses an interesting research challenge.

3. RESEARCH OPPORTUNITIES

Dava visualization is part of a larger data analysis process. Although we have proposed techniques for a DVMS to manage the data transformation, layout, and rendering processes for creating static data visualizations, our vision is for an interactive DVMS system that manages how data is viewed, explored, compared and finally published into stories for consumers to experience.

To this end, there are numerous interesting research opportunities to explore, such as (1) expanding our language proposal (Section 2.1) into a comprehensive language that can also describe user interactions in a manner that is amenable to cost-based optimization, (2) understanding interaction and visualization-specific techniques that can be used in an optimization framework to either meet interactive (100ms) latency constraints or mask high-latency queries, (3) exploiting different classes of hardware (e.g., GPUs) that are optimized for specific types of visualizations, and (4) incorporating recommendation and higher-level analysis tools that help users gain *sound* insights about their data.

The rest of this section outlines some immediate steps that help address each of these research directions.

3.1 Visualization Features

Lineage-based Interaction: Brushing and linking is a core interaction technique (Figure 1 arrow 6) where the user selects data in one view, and manipulates (e.g., highlights, removes) the corresponding data in other views. To do this, selected elements must be traced back to their input records, and then forward from those inputs to visual elements in the other views. Unfortunately, existing visualization tools either require users to track these lineage relationships manually [7], or provide implementations that often scale poorly to larger datasets and more complex visualizations.

In contrast, our relational formulation captures these lineage relationships automatically, and can thus express brushing and linking as lineage queries. Furthermore, workflows allow the DVMS to optimize and scale interactions to very large datasets with little user effort. For example, Ermac can automatically generate the appropriate data cubes and indices to optimize brushing and linking similar to the techniques used in imMens [20] and nanocubes [19].

Although the database community has explored many lineage optimizations [33, 13, 17, 11], additional techniques such as pre-computation and approximation will be necessary for supporting an interactive visualization environment.

Visualization Estimation and Steering: Users can easily build workflows that execute slowly or require significant storage space to pre-compute data structures, and it would be valuable to alert users of such costs. The DVMS can make use of database cost estimation [26, 8, 9] techniques to inform users of expensive visualizations (e.g., a billion point scatterplot) and inherent storage-latency trade-offs, and to steer users towards more cost-effective views. The latter idea (e.g., query steering [2]) may benefit from understanding the specification that produced the queries.

Rich Contextual Recommendations: Recommending relevant or surprising data is a key tool as users interactively explore their datasets. Prior work has focused on recommending visualizations and queries based on singular, but semantically different features such as data statistics [21], image features [23], or historical queries [18, 24, 25]. Ermac controls, and can thus use, all of these features to construct more salient recommendations to the user. For example, image features such as mountain ranges may be of interest when rendering maps, whereas the slope of a line chart is important when plotting monthly expense reports.

Result analysis: Several recent projects [22, 31], including one of the author's Scorpion project [32], extend databases to automatically *explain* anomalies and trends.Thus the DVMS can use these extensions "for free" to not only *present* data, but also embed functionality to automatically *explain and debug* the results.

3.2 Query Execution

Developing visualizations that are interactive across various environments and client devices (e.g., phone, laptop) can be challenging. Ermac can allow users to specify latency goals (e.g., 200ms interaction guarantees) and use **Rendering Placement** and **Psychophysical Approximation** optimizations to satisfy these constraints.

The former dynamically decides where to render visualizations given the client's available resources. For instance, heatmaps may be faster to render server-side and send to the client as a compressed image, whereas histograms are faster to send as data records and render on the client.

The latter produces approximations in a way that minimizes user perceived error, and is widely used in image and video compression. For example, humans are sensitive to position but have trouble discerning small color variations. Ermac can then respond to poor network bandwidth by pushing down an aggregation operator to coarsely quantize the color of a heatmap to match a smaller data type (e.g., `short` instead of `long`), and thus reduce the bandwidth demand by 4×. Alternatively, Ermac can aggregate the histogram data into coarse bins and use pre-computed data structures to reduce latency. Developing sufficient annotations to automate this optimization is an interesting research direction.

Finally, Ermac can use **Occlusion Filtering** to minimize unnecessary work. A common technique in computer graphics is visibility culling [10], which filters geometric objects that are hidden behind closer objects. While these optimizations are readily applied when rendering the `data` relation containing geometries, Ermac may be able to apply these *occlusion filters* as data-space transformations earlier in the LVP to avoid generating occluded visual elements. For example, a plot that layers a histogram over a heatmap can first render the histogram and push down a filter to remove data corresponding to the occluded heatmap pixels.

4. RELATED WORK

Previous work in visualization systems have traded-off between expressiveness and performance. For instance, popular toolkits such as D3 [7], protovis [6] and matplotlib [12] are highly expressive, however they require low level programming that impedes the ability to quickly iterate and do not scale to large datasets. Declarative grammar-based languages such as the Grammar of Graphics [30] and ggplot2 [1] are expressive domain-specific languages designed for rapid iteration, however they do not scale beyond their host environments of SPSS and R.

Recent systems address these scalability limitations by either adopting specific data management techniques such as pre-computation [20], indexing [19], sampling [4], speculation [15], and aggregation [5, 29], or developing two-tiered architectures where the visualization client composes and sends queries to a data management backend [27, 14]. The former approaches are optimized towards properties of specific applications or visualization types and may not be broadly applicable. The latter approach forgoes the numerous cross-layer optimizations described in this paper.

Our proposed Ermac DVMS is intended be both *expressive* thanks to the declarative visualization language and *performant* by using traditional database optimizations as well as those outlined in Section 3.

5. CONCLUSIONS

The explosive growth of large-scale data analytics and the corresponding demand for visualization tools will continue to make database support for interactive visualizations increasingly important. We proposed Ermac, a Data Visualization Management System (DVMS) that executes declarative visualization specifications as a series of relational queries, and explored several challenges and optimization opportunities for the future.

6. REFERENCES

[1] ggplot2. `ggplot2.org`.

[2] *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

[3] The lyra visualization design environment (vde), February 2014. `http://idl.cs.washington.edu/projects/lyra/`.

[4] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. 2013.

[5] L. Battle, R. Chang, and M. Stonebraker. Dynamic reduction of query result sets for interactive visualization. 2013.

[6] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *InfoVis*, 2009.

[7] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *InfoVis*, 2011.

[8] S. Chaudhuri, V. Narasayya, and R. Ramamurthy. Estimating progress of execution for sql queries. In *SIGMOD*, 2004.

[9] S. Chaudhuri and V. R. Narasayya. Autoadmin 'what-if' index analysis utility. In *SIGMOD*, 1998.

[10] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computers*, 2003.

[11] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *KDD*, 1997.

[12] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 2007.

[13] R. Ikeda and J. Widom. Panda: A system for provenance and data. *IEEE Data Eng. Bull.*, 2010.

[14] J.-F. Im, F. G. Villegas, and M. J. McGuffin. Visreduce: Fast and responsive incremental information visualization of large datasets. In *BigData Conference*, 2013.

[15] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and Interactive Cube Exploration. In *ICDE*, 2014.

[16] S. Kandel, A. Paepcke, J. Hellerstein, and H. Jeffrey. Enterprise data analysis and visualization: An interview study. *VAST*, 2012.

[17] A. Kemper and G. Moerkotte. Advanced query processing in object bases using access support relations. In *VLDB*, 1990.

[18] A. Key, B. Howe, D. Perry, and C. R. Aragon. Vizdeck: self-organizing dashboards for visual analytics. *SIGMOD*, 2012.

[19] L. D. Lins, J. T. Klosowski, and C. E. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 2013.

[20] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. *EuroVis*, 2013.

[21] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 2007.

[22] A. Meliou, W. Gatterbauer, and D. Suciu. Reverse data management. *PVLDB*, 2011.

[23] A. Oliva and A. Torralba. Building the gist of a scene: the role of global image features in recognition. In *Progress in Brain Research*, 2006.

[24] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina. Seedb: Visualizing database queries efficiently. *PVLDB*, 2014.

[25] S. Sarawagi and G. Sathe. i3: Intelligent, interactive investigaton of olap data cubes. In *SIGMOD*, 2000.

[26] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, 1979.

[27] C. Stolte and P. Hanrahan. Polaris: A system for query, analysis and visualization of multi-dimensional relational databases. *InfoVis*, 2002.

[28] R. Wesley, M. Eldridge, and P. T. Terlecki. An analytic data engine for visualization in tableau. In *SIGMOD*, 2011.

[29] H. Wickham. Bin-summarise-smooth: a framework for visualising large data. Technical report, had.co.nz, 2013.

[30] L. Wilkinson. *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc., 2005.

[31] W. Willett, J. Heer, and M. Agrawala. Strategies for crowdsourcing social data analysis. In *CHI*, 2012.

[32] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 2013.

[33] E. Wu, S. Madden, and M. Stonebraker. Subzero: A fine-grained lineage system for scientific databases. In *ICDE*, 2013.