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 – the visualization tool often reimplements many data processing operators – 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. In this way, the DVMS is both *expressive*, thanks to the declarative visualization language, and *performance*, because it can take advantage of traditional and novel visualization specific optimizations to scale interactive visualizations to massive datasets.

1. INTRODUCTION

The holy grail of visualization systems is one that lets users explore different facets of the data so easily, and recommends views based on the user's interaction patterns that are so relevant, that users rapidly converge onto valuable insights – irrespective of the size of the data. Unfortunately, existing visualization systems fall far short of this goal.

Most visualizations today 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., only retrieve data within the a visible bounding box), visualization tools often try to avoid excessive roundtrips to the database by managing their own cache of results, and executing data transformations directly (i.e., outside of the database).

This two-tiered approach has three significant drawbacks. First, the database is not aware that different queries are related and may repeatedly recompute the same results. For example, slightly panning a map visualization will issue a query to recompute the entire map, though most of the results have not changed. Second, current visualization tools attempt to re-implement basic database functionality, including grouping and filtering operations, as well as computing statistical summaries. Some tools even go so far as implenenting an entirely new database for this purpose [29]. Lastly, in contrast to many modern databases, the overwhelming majority of visualization tools assume that datasets fit entirely in memory (e.g., D3 [8], R and Matlab). The afore-mentioned duplicate functionality is built on this assumption, making it extremely inefficient or inapplicable for massive datasets, which rarely fit in memory.

We propose instead to integrate these two systems into a *Data Visualization Management System* (DVMS), which can take full advantage of all features provided by the database. A DVMS easily scales common data transformations to massive datasets by executing these operations directly in the database. Furthermore, a DVMS can leverage additional database functionality implement interactivity with little additional effort. For example, if the database engine supports lineage queries, we can automatically link related geometric objects (e.g., circles, rectangles) across views by tracking overlap between the input records that generated them. An integrated design can also take advantage of novel *visual optimizations* to reduce rendering latencies, such as: (1) applying occlusion rules to filter out records that generate geometric objects that end up hidden from the user's view; (2) output-based downsampling of datsets to scale with the size of the viewport; and (3) assigning rendering operations between the client and backend based on current computational load, required resources, and network congestion.

We present the design methodology for our integrated DVMS Ermac, which consists of two major steps: (1) users specify a complete visualization workflow (i.e., mapping from datapoints to geometric primitives on the screen) using a declarative visualization language; and (2) we compile the resulting workflows into SQL queries, which are executed in the underlying DBMS. Using this design, we can leverage traditional database optimizations to improve rendering and processing performace, 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, 31]. We now describe how a visualization specification is first represented as

a Logical Visualization Plan (LVP) that is further 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 dataset will be an `election` table containing Obama and Romney's campaign expenditures during the 2012 US presidential election. The attributes of the table include the candidate name, their party affiliation, the purchase dates within a 10 month period (Feb. to Nov. 2012), the amount spent, and the recipient:

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

2.1 Logical Visualization Plan

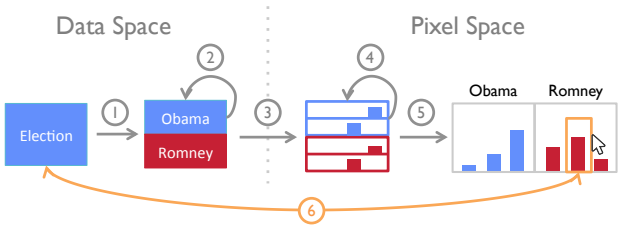

Figure 1: Logical Visualization Plan for `expenses` example.

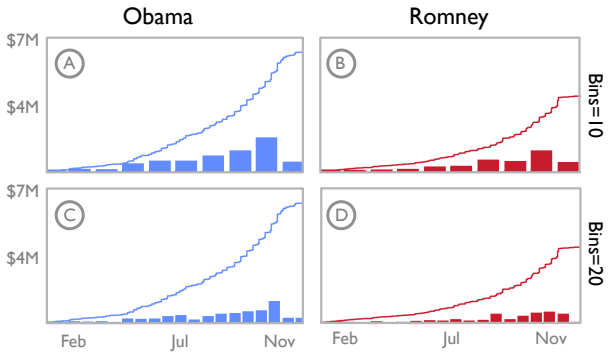

Figure 2: Faceted visualization of `expenses` table

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)]
```

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 1,2), pixel space (arrows 4,5), or between the two (arrow 3). This subsection describes these five classes in further detail using an example that visualizes 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
]*
```

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 maps 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.

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. For example, to produce the above histogram, Ermac first

[1]The PVP will be further optimized and compiled by the DBMS into a traditional physical *query* plan.

[2]layers are listed in drawing order

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 ($[Feb, Nov]$) pixel coordinates ($[0, 100]$) along the x 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 unnecessary replication when executing the cross-product. Identifying further optimizations for individual and across multiple LVP operators poses an interesting research challenge.

3. OPTIMIZATION OPPORTUNITIES

We now present examples of interaction and visualization features that are made possible by Ermac's visualization and data processing integration, and execution optimizations that take advantage of semantics inferred from the visualization specification.

3.1 Visualization Features

Lineage-based Interaction: Brushing and linking [6] is a core interaction technique (Figure 1 arrow 6) where the user selects data in one view, and highlights, removes, or otherwise manipulates the corresponding data in the other views. To do this the selected elements need to be traced back to their input records, and forward from those inputs to the visual elements in the other views. Unfortunately, existing visualization systems either expect the user to manually track how each record is transformed and aggregated into the final visual elements [8, 28], or provide pre-packaged implementations that often scale poorly to larger datasets and more complex visualizations.

In contrast, our relational formulation captures these input-output relationships (lineage) automatically and can thus express these interactions as lineage queries over these relationships. This declarative specification lets the DVMS optimize and scale visualization interactions to very large datasets with minimal 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 [19] and nanocubes [18].

Although the database community has explored many lineage optimizations [34, 14, 16, 12], additional techniques such as pre-computation and approximation will be necessary for supporting an interactive visualization environment.

Visualization Estimation and Steering: Users can easily write a specification that executes very slowly or requires significant storage space to pre-compute data structures, and it would be valuable to alert users of such costs. Our relational algebra formulation can make use of database cost estimation [26, 9, 10] techniques to inform users of hard to render visualizations (e.g., explicitly rendering a billion point scatterplot) and inherent storage-latency trade-offs, and to steer users towards views that can be rendered efficiently. The latter idea was recently explored in the context of database query steering [2] and may benefit from understanding the specification that produced the queries.

Rich Contextual Recommendations: Recommending relevant or surprising data and visualizations is a key tool as users interactively explore their datasets. Prior work has focused on developing visualization and query recommendations based on singular features such as data statistics [20], image features [23], or historical queries [17, 24, 25]. Ermac controls, and thus can use, features across multiple semantic levels – data statistics, historical queries, visualization trends, and pixel features – to construct more salient recommendations to the user. For example, when rendering geographic data, image features such as mountain ranges may be of interest , whereas the slope of a line chart is important when visualizing monthly expense reports.

Result analysis: Several recent projects [21, 32], including one of the author's Scorpion project [33], extend databases to *explain* anomalies and trends in query results by generating explanations automatically, or asking crowd workers. As we continue to develop these extensions, the DVMS can use them "for free" to not only *present* data, but also automatically embed interactive functionality to *explain and debug* the results.

3.2 Query Execution

It is difficult to develop a visualization that is interactive across different network connectivities and client devices (e.g., phone, laptop). Ermac can extend its declarative

language to allow users to specify latency goals (e.g., interactions should take less than 200ms) and use **Rendering Placement** and **Psychophysical Approximation** optimizations to satisfy these latency constraints.

The former dynamically decides where to render the visualization depending on the client's available resources. For instance, a heatmap may be more efficiently rendered on the server and sent to the client as a compressed image, whereas a binned histogram or choropleth is faster to transmit as data records and render on the client.

The latter exploits human psychophysical limitations (e.g., humans are very sensitive to position but much less sensitive to small color variations) to preferentially approximate values in ways that minimize user perceived error – these types of techniques are widely used in image and video compression. For example, Ermac may respond to poor network bandwidth by pushing down an aggregation operator to coarsely quantize the color of a heatmap so that it can be represented as a `short` instead of a `long`, and thus reduce the bandwidth demand by 4×. Alternatively, Ermac can aggregate the histogram data into coarse bins and use precomputed data structures to speed up the queries. Developing the 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 [11], 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 geometry records in the first place. For example, a graphic that layers a histogram on top of a heatmap can first compute the histogram layer and push down a filter operator that removes data corresponding to the occluded heatmap pixels.

3.3 Broader Challenges

Our prosposed work reveals three major research challenges for future work in interactive visual exploration of massive datasets. First, the query latencies explored in current database visualization research (seconds to minutes) is orders of magnitude behind the latency guarantees assumed by the information visualization and graphics communities (milliseconds). Furthermore, it has been observed that significant latencies (greater than 100ms) between interactions directly affects user behavior as they visually explore their data [**?**]. Moving computation to client/server nodes to reduce idle-time and leveraging visualization-specific reductions/filters are short-term attempts to address this issue. How can we update existing database functionality to achieve truly interactive latencies for the purpose of visual exploration?

Second, there is currently an unavoidable tradeoff between storage costs and online performance (i.e., speed/latency) when using pre-computed reductions (e.g., aggregation, sampling) to limit the amount of data that must be processed at runtime. Revealing this tradeoff to users gives them the ability to choose more efficient paths for themselves, but does not remedy the issue. There are future opportunities here to produce better reductions that more efficiently utilize the underlying storage, while maintainig or reducing online latency when producing interactive visualizations in real time.

Lastly, user interactions are currently only integrated client-side in visualization tools, and databases are completely oblivious of this functionality. As mentioned previously, this leads to missed opportunities for significant server-side performance improvements. We have proposed one approach to incorporating brushing and linking as a first-class operation in our integrated DVMS, but have not addressed the countless other operations available to users in most visualization systems. Comprehensive extensions to database query languages (and thus query optimizers) to fully integerate user interactions is absolutely necessary to provide effective DVMS's.

4. RELATED WORK

Previous work in visualization systems have traded-off between expressiveness and performance. For instance, popular toolkits such as D3 [8], protovis [7] and matplotlib [13] 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 [31] 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 [19], indexing [18], sampling [4], and aggregation [5, 30], or developing two-tiered architectures where the visualization client composes and sends queries to a data management backend [27, 15]. 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 need for visualization tools usable by both data scientists and enthusiasts [22] will continue to make database support for interactive visualizations more and more 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] R. A. Becker and W. S. Cleveland. Brushing scatterplots. *Technometrics*, 1987.

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

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

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

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

[11] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER*, 9(3), 2003.

[12] 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.

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

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

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

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

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

[18] L. D. Lins, J. T. Klosowski, and C. E. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Vis. Comput. Graph.*, 19, 2013.

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

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

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

[22] K. Morton, M. Balazinska, D. Grossman, and J. D. Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *PVLDB*, 2014.

[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. 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] C. Weaver. Building highly-coordinated visualizations in improvise. In *INFOVIS*, 2004.

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

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

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

[32] W. Willett, J. Heer, and M. Agrawala. Strategies for crowdsourcing social data analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI, 2012.

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

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