Managing Data for Visual Analytics: Opportunities and Challenges

Jean-Daniel Fekete
INRIA Saclay

Claudio Silva
NYU Poly and NYU CUSP

Abstract

The domain of Visual Analytics has emerged with a charter to support interactive exploration and analysis of large volumes of (often dynamic) data. A common feature shared by all the visual analytics applications developed so far is the reliance on ad-hoc and custom-built mechanisms to manage data: they re-implement their own in-memory databases to support real-time display and interactive feedback and analytical algorithms (e. g., clustering, multidimensional projections, specific data analyses) to overcome the delay required to exchange data with specialized analytical environments, such as Matlab, R, and the myriad of more specialized systems and command-line programs. This article presents visual analytics scenarios requiring a new generation of databases to support and foster the new era of interactive analytical environments. This new generation would relieve visualization researchers from sub-optimally re-implementing database technologies. We argue that the new services required to support interactive explorations present research challenges to the database community and can increase the utility and simplicity of integration of the new generation of databases for data-intensive applications.

1 A Fresh Look at Large Data Visualization

In database research, the big data issue has mainly been addressed as a scale issue: storing and providing the same level of services as before in terms of speed, reliability, interoperability and distribution. The scale challenges are now being solved by research and industry. A less advertised issue raised by the increase of available data is the unprecedented opportunity for discoveries and exploratory studies. For example, Metagenomics is a research field in Biology that sequences DNA from random samples of material collected in the wild to discover the largest possible diversity of living species. With new high-throughput sequencing technologies, the genome of millions of species are sequenced and stored in databases but are not systematically explored due to the lack of appropriate tools. Bank transactions are being logged and monitored to find patterns of frauds, but new schemes arise continually that need to be discovered in the billions of transactions logged daily. This task of discovery of innovative fraud schemes is not well supported by existing tools either. Similar problems arise in a wide range of domains where data is available but requires specific exploration tools, including sentiment analysis on the Web, quality control in Wikipedia, and epidemiology to name a few.

However, current database technologies do not meet the requirements needed to interactively explore massive or even reasonably-sized data sets. In this article, we discuss what the requirements are, why they are not

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

met and, in some cases, what could be done to better support them. Interactive exploration of large data is the goal of the new multidisciplinary field of Visual Analytics (VA), which brings together *visualization*, *interactive data analysis* and *data management*.

Visualization uses the human vision system to understand and find insights on data by displaying it with an adequate visual representation. The visual capabilities of humans far exceed their symbolic capabilities for processing a large number of items: exploring data tables containing thousands of items with tens of dimensions (e. g., all the available models of cameras on the market today with their main features) is a routine task for a modern information visualization (infovis) application but would be very demanding by merely looking at the same data with a standard spreadsheet. However, infovis is limited to the human visual perception that work well with thousands of data items and can, in some specific cases, explore data sets with about 1 million items, but certainly not above. Existing data sets easily exceed this number and still need to be explored. This is why VA combine analytical algorithms in the exploration process to overcome the dimensionality problems. Analytical algorithms can reduce the number of items (e. g., by sampling or aggregating), can reduce the number of dimensions (e. g., by projecting data, or finding dependencies between dimensions), can extract more meaningful items (e. g., by feature extraction) and can help find trends, patterns and outliers. One important challenge when connecting analytical algorithms with interactive visualization is to keep the system interactive and reactive. Most analytical algorithms available off the shelf are meant to process data from start to end regardless of the time it takes. Humans conducting interactive tasks expect results to appear quickly, even if incomplete initially or only rough estimates. Turning the existing analytical algorithms into reactive any-time algorithms is one important challenge that visual analytics needs to address.

In addition to the intrinsic running time of these algorithms, data exchange is currently a bottleneck. Most analytical environments—whether generic such as R or Matlab, or more specific—read the input data from files or databases, perform their computations and write their results in output files or databases. For large data sets, the input/output time exceeds by far "interactive time", the time required to perceive a program as reactive or interactive. To be effective, interactive exploration implies system requirements that are unusual for data management and are therefore not provided in standard databases, forcing VA developers to re-implement their data storage services in an ad-hoc way, usually sub-optimal for the standard services, and hampering interoperability.

We will discuss this issue in more detail in the next section, where we describe systems designed to support visual data exploration and discuss their requirements and limitations regarding data management. We also discuss real applications (UV-CDAT and Wikipedia) that illustrate needs and limitations of standard data management systems. The last section summarizes the opportunities and challenges in light of the requirements outlined before.

2 Visualization Systems

To turn data into visual depictions and provide interactions to navigate, analyze and explore them, visualization needs software libraries that load data, apply transforms through a pipeline of operations to eventually draw the results on a screen and manage input devices to interpret interactions. The two main sub-fields of visualization have slightly different architectures that are described below, with data management issues highlighted.

2.1 Scientific Visualization

The most common data types in scientific visualization systems come from a relatively small set, and can be described succinctly. The data has a given dimension (mostly 1D, 2D and 3D) and data sets are composed of very large collections of cells. The geometry of a cell is determined by its corresponding nodes, and the way the cells are glued together determine their topological structure. Besides geometric properties, there is often associated data (e. g., temperature, wind velocities). Explicitly storing nodes, cells, and their relationships

can be wasteful, in particular since much data belong to well-defined categories that can be stored much more efficiently. Scientific visualization systems tend to have a collection of pre-determined data formats that can efficiently store certain types of classes of data. This has really been a major requirement in the area given the large sizes of the data, and the need to support interactivity.

Take for example the data sets supported by VTK, one of the most widely used and comprehensive visualization toolkits. VTK supports a collection of structured and unstructured data sets. (We are not being comprehensive here, and we refer the extensive documentation of VTK for details.) Structured data sets are n-dimensional matrices, mostly images and volumes. For these data sets the topology is intrinsic, and many basic data operations (e. g., cell location, data look up) are very efficient. Unstructured data sets, where the nodes and cells need to be explicitly defined come in multiple flavors, mostly due to efficiency reasons. A triangulated surface is essentially a version of a 2D surface embedded in 3D space, and it is an example of an unstructured grid, since it requires explicit cell topology information.

Although in principle scientific visualization systems can use existing database technology, the reality is that they implemented all their own (rudimentary) data models, file formats, and metadata management. In the late 1990s, in order to handle ever larger data sets,[1] techniques from the database community started being used in visualization systems. Overviews of those efforts have been given in courses and tutorials by Cox and Ellsworth [8] and Silva et al. [21].

It is interesting to look at the internal processes and data structures used by visualization systems. Scientific visualization systems have traditionally adopted a "data flow" model, where data manipulation operations are represented by linked modules with typed inputs and outputs. These modules define the computations to transform data, and eventually generate pictures. Although conceptually simple, executing such networks gets complicated, in particular when data are large. Optimizations are necessary to avoid wasteful computations under the dataflow model, and caching and parallelism are often used (see, e. g., Bavoil et al [1] and Childs et al [6]). Data sets have temporal components and are so large as to require streaming and out-of-core processing, which further complicate efficient execution of the pipelines. In order to support interactivity, level-of-detail algorithms, where approximate data flows through the module network, need to be supported [22].

While database systems could support many data management tasks for visualization system, such as metadata management, indexing, caching, etc., they have not been widely used. Part of the problem comes from the lack of database expertise by visualization experts. Also, there appears to be a "gap" in functionality needed to efficiently implement visualization algorithms that work on data stored in a database system. For example, support for efficient, vendor-neutral description of multidimensional geometry and related fields needs to be available, as well as for primitives that support level-of-detail (approximate) queries in rendering algorithms.

2.2 InfoVis Systems

Popular infovis toolkits such as Prefuse [12] or the InfoVis Toolkit [9] implement an in-memory database as their main data structure. They transform data tables into visual tables, containing mostly graphical attributes—such as geometric shape, position, color— and render them on the screen.

In InfoVis applications, visual tables are used manage rendering. These in-memory data tables are column-oriented, so adding or removing a column is very cheap and memory access is optimized when few columns are read out of several, which is typical in visualization. The performance requirements are stringent since these tables are used to render the visualization at an interactive rate, typically around 60-100 times a second (60-100 Hz). In current implementations, the data table query is not the bottleneck for rendering large data sets, instead, the bottleneck is the graphics engine. The database constraints are not only in the throughput but also on the latency, because the rendering should be done in sync with the refresh rate of the screen—so the visual table data must be available in main memory all the time.

[1] One article traces the first use of the term "Big Data" to a visualization paper by Cox and Ellsworth.

There are several reasons why the InfoVis community would like to rely on standard database engines instead of the ones they implemented: the services offered by infovis in-memory databases are very rudimentary compared to the standard database services, they need to access standard databases, forcing infovis developers to use two slightly different programming interfaces, which is cumbersome and more complex. However, there are several reasons why standard databases cannot be used currently. We believe the particular services required by the visualization community would benefit many other communities and they may also raise interesting challenges to the database community.

Even if standard database technology would provide optimized library to access databases from regular applications—which they often do—there is currently no way to specify what portion of a queried table should remain in main memory so as to fulfill the real-time constraints. The data types supported by existing databases is not suitable for visualization for two reasons. First, SQL column types need to encode data semantics to allow the selection of good default visual representations. Second, visualization needs to manage geometry in visual tables, and there is currently no support for that. The GIS community has defined some types of standard geometries for their needs, but they have been designed to encode exact 2D topology for mapping management instead of compact geometry for fast rendering. Better support for geometry would be important, both in 2D and 3D, in particular to manage box queries (e. g., what are the objects that are visible in a given box?).

Notification is used extensively in visualization to support linked views and brushing. When data tables are changed, the related visual tables are recomputed, and when the visual tables are changed, the screen is updated. These modifications can come from data updates but are more frequently caused by interactions. For example, clicking on an item on screen will select an object through a selection column in the visual table. This selection will also create a new graphical entity to highlight the selected item, which will be rendered again on screen. The selection can also be managed at the data table level, which is very common when several visual tables are managed for a single data table, i. e., the data table is visualized with multiple visual representations (e. g., showing a spreadsheet view and a scatterplot view). InfoVis databases provide a simple "trigger" mechanism that perform this notification. Implementing a notification mechanism on top of a regular database is complicated and not portable. Client-side notification should be a standard service.

2.3 Visual Analytics

Combining visualization with interactive analysis to explore large data set is the goal of Visual Analytics. The need to support analytical computations creates additional challenges. For example, an infovis system such as Prefuse can load a table from an flat-file format it supports (e. g., a CSV table), pass it to a clustering module to compute groups of items (e. g., using the Weka machine learning toolkit), reduce the table by aggregating the groups and visualize the results (e. g., using a scatterplot visualization). The standard method to implement this pipeline is to copy the content of a Prefuse table into the data model of the clustering program, then to read back in Prefuse the resulting table with one column added, containing a cluster number for each row to be able to project and visualize it. Two data copy operations are thus needed, usually through files.

To address some of these issues, we have designed an abstraction on top of in-memory data tables in a system called Obvious [10]. We have implemented several bindings to check its generality, applicability, and measure its overhead. As we expected, abstracting-out the data model of infovis and data-mining toolkits allows combining very efficiently all these software packages, cutting down the overhead of memory copy whenever possible. The services offered by the Obvious toolkit are a subset of the services we expect from a modern database. Extensive cache management to move data in the memory of the applications that will need it, notification at the client side to manage propagations of value changes, and chains of computations for dynamic data. Propagating value changes from cache to cache is an extremely important operation for VA. In some cases, the visualization and analysis modules can share the same memory (e. g., Java Virtual Machines memory) but, in more realistic settings, the memory cannot be shared because some applications use a specific memory layout that is not abstracted, or because the application needs to run on a different computer (e. g., in the Cloud). In these

cases, the data should move from one memory to the other, using a distribution mechanism. Currently, this communication needs to be managed explicitly by each VA application or, if a distributed database is used, through the database. However, the work consisting in collecting the data changed from one memory to the other, when managed by the VA application, is sub-optimal. Indeed, if the database manager was managing a replicated cache, it would know what portion of the data has been altered and would only move that part. Optimizing that smart distribution is beyond the skills of a VA application programmer. However, from inside the database engine, it looks a lot like database replication and is well understood—though complex to implement correctly.

3 Workflows for Visual Analysis

Visualization systems are just components required for Visual Analytics. Combining computation tools, visualization, and extraction/selection from databases is necessary and can be implemented using ad-hoc tools or more generic *workflow* systems. We report on two of them to discuss the requirements they have on data management services.

3.1 VisTrails

VisTrails (www.vistrails.org) is an open-source provenance management and scientific workflow system that was designed to support the scientific discovery process. VisTrails provides unique support for data analysis and visualization, a comprehensive provenance infrastructure, and a user-centered design. The system combines and substantially extends useful features of visualization and scientific workflow systems. Similar to visualization systems [15, 17], VisTrails makes advanced scientific visualization techniques available to users allowing them to explore and compare different visual representations of their data; and similar to scientific workflow systems, VisTrails enables the composition of workflows that combine specialized libraries, distributed computing infrastructure, and Web services. As a result, users can create complex workflows that encompass important steps of scientific discovery, from data gathering and manipulation, to complex analyses and visualizations, all integrated in one system.

Whereas workflows have been traditionally used to automate repetitive tasks, for applications that are exploratory in nature, such as simulations, data analysis and visualization, very little is repeated—change is the norm. As a user generates and evaluates hypotheses about data under study, a series of different, albeit related, workflows are created as they are adjusted in an iterative process. VisTrails was designed to manage these rapidly-evolving workflows. Another distinguishing feature of VisTrails is a comprehensive provenance infrastructure that maintains detailed history information about the steps followed and data derived in the course of an exploratory task [20]: VisTrails maintains provenance of data products (e. g., visualizations, plots), of the workflows that derive these products and their executions. The system also provides extensive annotation capabilities that allow users to enrich the automatically captured provenance. This information is persistent as XML files or in a relational database. Besides enabling reproducible results, VisTrails leverages provenance information through a series of operations and intuitive user interfaces that aid users to collaboratively analyze data. Notably, the system supports reflective reasoning by storing temporary results, by providing users the ability to reason about these results and to follow chains of reasoning backward and forward [18]. Users can navigate workflow versions in an intuitive way, undo changes but not lose any results, visually compare multiple workflows and show their results side-by-side in a visual spreadsheet, and examine the actions that led to a result [20, 1]. In addition, the system has native support for parameter sweeps, whose results can also be displayed on the spreadsheet [1].

VisTrails includes a number of components that might be best supported directly in a database management system. For instance, VisTrails contains a number of abstractions for files, which are supposed to make it easier to abstract the file type, location (local or Web), and to perform version management. One example is the "persistent file" concept[16], which enables input, intermediate, and output files to be managed through a

version control system (e. g., git). The VisTrails workflow execution engine is user extensible, and it is able to support caching and complex dependencies. In an ideal world, the system would build on state kept by a database. One step towards the use of database technology is a recent addition to the system where it is now possible connect to a provenance-aware relational database [7]. When used correctly, all this functionality nicely complements VisTrails workflow provenance to provide complete, end-to-end provenance support.

3.2 EdiFlow

EdiFlow is an experimental workflow system designed for VA [2] with a focus on reactive computation. A workflow is specified over a database as a set of modules computing the values of output tables from input tables. It is reactive because when a table is modified, the recomputation of the dependent modules is triggered, changing some output tables that can be the input tables of other modules, until all the output tables are recomputed. An important issue for EdiFlow is to avoid useless computations by providing details to modules of what changed in its input tables (the changes to a table are called its delta) so that it can optimize the recomputations. The changes are supposed to arrive by chunk and not one item at a time so the reconciliation are typically managed within transactions. However, since changes can occur at any time, modules can be notified that some input value has changed before their computation is finished. In that case, modules can choose various strategies according to their semantic and implementation. For example, they can stop immediately their computation to restart them from scratch at the next recomputation with the new values. They can also decide to reconcile the new values right away. All these mechanisms are meant to handle database changes asynchronously. These changes can occur for two reasons: either because some data source is changing (e. g., dynamic data is coming in), or because some user has changed values during an interaction with the system. This is the case when EdiFlow is connected to a visualization system and an operation has been performed, such as a user selecting several visualized items, that change the values of a "selection" column in a visual table, that in turn triggers a recomputation of the visualization. While VisTrails manages very effectively the interactive management of scientific workflows, EdiFlow's strength lies in its ability to perform continuous computation of dynamic data.

The heart of EdiFlow relies on two mechanisms that are not standard in databases: notification management (or *client triggers*) and isolation of table values for module computation. The first mechanism is required to keep track of which module should be executed when tables are changed. Each module using an input table that is modified is flagged for execution or recomputation. In addition, for recomputation, each module should be provided the delta, which is maintained by EdiFlow using the trigger mechanism. The second mechanism is required to avoid input tables from a module being run, to be changed spuriously by another module. We have to manage timed values in EdiFlow databases to implement this isolation.

4 Use Cases / Example Apps

We give two very different examples to help understand VA and its data management needs.

4.1 Climate data analysis (UV-CDAT)

Climate scientists have made substantial progress in understanding Earth's climate system, particularly at global and continental scales. Climate research is now focused on understanding climate changes over wider ranges of time and space scales. These efforts are generating ultra-scale data sets at very high spatial resolution. An insightful analysis in climate science depends on using software tools to discover, access, manipulate, and visualize the data sets of interest. These data exploration tasks can be complex and time-consuming, and they frequently involve many resources from both the modeling and observational climate communities.

Because of the complexity of the explorations, the number of tools, and the amount of data involved, database support, including provenance, is critical to support reproducibility and allow scientists to revisit existing com-

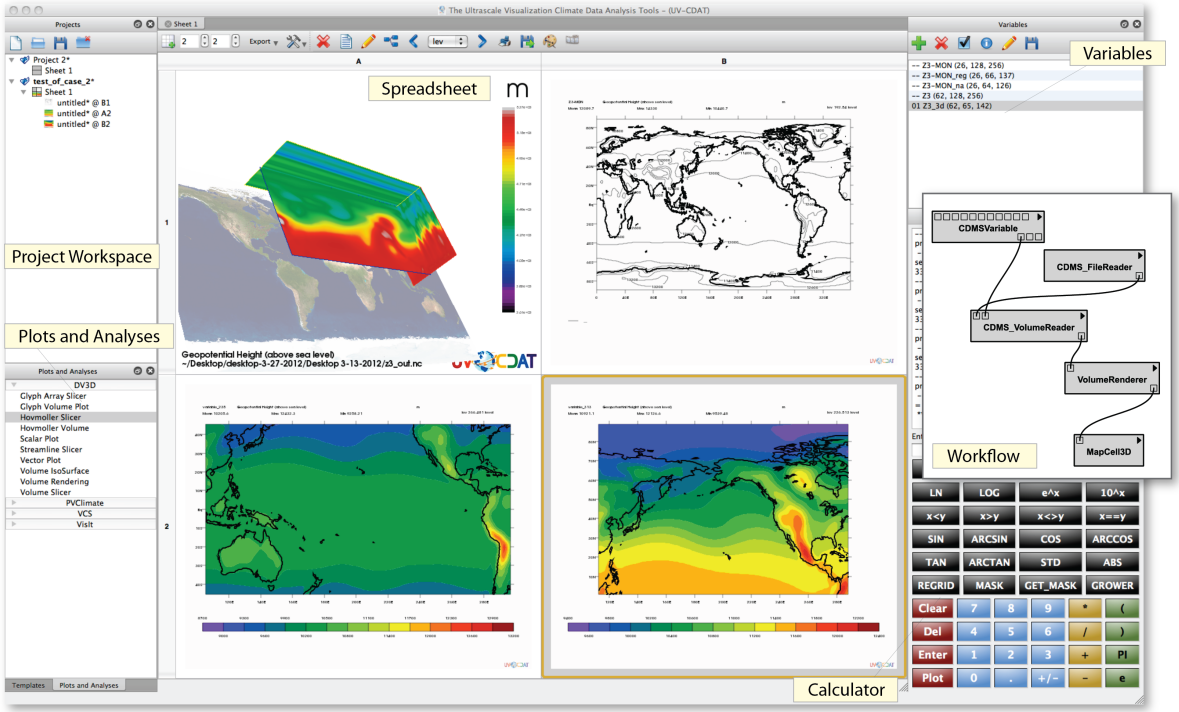

Figure 1: UV-CDAT GUI. Spreadsheet (middle), Project View (top left), Plot View (bottom left), Variable View (top right), and Calculator (bottom right). The system aims to streamline visual data analysis by the use of advanced workflow and provenance support. Image courtesy of Emanuele Santos.

putational pipelines, and more easily share analyses and results. In addition, as the results of this work can impact policy, having provenance available is key to support decision makers. UV-CDAT [19] is a provenance-aware visualization system aimed at supporting large-scale climate data analysis. It uses the VisTrails SDK, which brings data management and provenance capabilities (see Figure 1).

To highlight how UV-CDAT might be used, we use a simplified example from Jerry Porter, a scientist at NASA. The scientist is looking at data from paleoclimate runs on the CCSM3. He wants to determine if the variance of the DJF (December-January-February average) 500 hPa heights (the level of the 500 millibar pressure surface) changes from two different paleoclimate climate simulations. This should give an indication of the changing location of storm track and could be a test of what happens to extratropical storm tracks in a warming earth. He will also need to perform the same analysis for many different periods in the past. The list of steps performed in the analysis are the following: 1) Data discovery: The metadata for the daily model output from the model runs are examined to find the variables. 2) Select a region of interest. For example, the West Coast of the US. 3) Pick a variable and run the variance calculation on the time dimension. 4) Save the data. 5) Plot a 3D Hovmoller diagram (latitude, longitude, time) using DV3D to see the time variation of the geopotential height. 6) Slice the data to examine the region of interest. 7) Plot 2D maps of the subregion, fix the color bar, overlay contours—experiment with other fields to overlay (e. g., moisture flux).

Normally, this process needs to be performed for several models, and the models compared. With existing tools, there is a huge reliance on "files" and naming conventions, and a lot of the meta data is not stored in interchangeable formats. Integration with database, workflow, and provenance systems would streamline the analysis and visualization process and make scientists more efficient.

4.2 Wikipedia Analysis

WikipediaVis [5] is a system to visually present indicators of quality for Wikipedia articles to help assess their trustworthiness. Computing some of these indicators is complex and time consuming. Since the Wikipedia foundation does not maintain computed indices, we decided to compute them on one of our machines and provide a web service to give access to them. We have designed a reactive workflow system to carry the computation task: WikiReactive [4]—the precursor of EdiFlow dedicated to Wikipedia. It computes a cascade of tables related to Wikipedia articles. All these computations are designed to enrich Wikipedia pages with analytical measures to help quickly assess their maturity, known to be correlated to their quality.

At the beginning of the cascade, a program polls Wikipedia to collect and store in our database the list of articles that need processing. Then, the workflow selects one of these articles, fetching its new contents from Wikipedia to compute several measures. For the sake of simplicity, let's assume we only want to compute and maintain, for each character of an article, the user who entered it. This information is very useful because it is the base of the computation of users quality and, accordingly, can be used to show what parts of an article are more trustworthy than others. The first step of our cascade consists in computing the differences (diffs) between the previous and the new version of the article. Other steps involve collecting per user statistics (e. g., number of edits, number of characters added taken from the diffs, etc.), computing the user-per-character table, then computing the real contributors of the article.

This workflow has been designed to gather data and compute measures dynamically and reactively when the pages change. It can be enriched with new modules if other measures become of interest for us, such as natural language analysis or sentiment analysis, thanks to our modular reactive architecture. This process can be seen as a step-wise enrichment of the primary data and is crucial to VA to support interactive search and navigation. For example, the computed diffs allow to quickly find who has inserted a specific word in a Wikipedia article, tremendously shortening the time to find authors of copyright infringements, but also to quickly look at the profile of user editions. This exploration would take hours if it had to be recomputed on the fly.

The data management issues in this application are similar to those faced with EdiFlow: *client triggers* and *isolation*. The main message is that VA applications need reactive workflows to compute important statistics, derive values, and analytical information from dynamic data sources.

5 Summary

Visualization has mimicked/reinterpreted database technologies for various reasons. We need to reconcile our software infrastructures for data management with standard databases but this requires some improvements and new capabilities.

Supporting geometric data used by visualization: visualization manages 2D and 3D data, VA needs to move this data across multiple execution environments. Standardizing geometry storage and supporting queries would address a major reluctance of visualization researchers towards using standard databases. GIS extensions already specify some geometry but not with the same semantics. For 2D visualization, the XML/SVG standard would fit the needs and is already well specified. For 3D, more work is needed to define vendor-neutral structures. In the latter case, a major issue would be storage efficiency since 3D geometry, as managed by current visualization applications, can be large.

Supporting geometry at multiple scales is also essential. Just like Google Earth specifies maps at different scales, visualization also needs to retrieve geometry at multiple scales. Geometric queries include intersection, and inclusion. Indexing these objects is a well-known issue in computational geometry and techniques from GIS databases can be reused.

Supporting extensive caching is mandatory to avoid duplicating data structures and maintaining the required computation and rendering speed for visualization structures. Interactive visualization needs high speed and low latency with real-time guarantees. We share the vision of Boncz et al. [3] of the database as a shared memory

with smart caching to use the existing hardware optimally. This vision differs from the classical database with explicit load and store operations. If in-memory caching is supported, *fast distribution of cached data at transaction time* should also be supported to propagate changes across applications. This would allow computation modules to exchange computed results quickly without having to implement their own communication mechanism when the database already knows how to distribute data. With extensive caching, *smart disconnection and reconnection management* should be supported to allow applications to run while connected but also in isolation.

For VA, *long transaction models* should be supported. More work is needed to define a minimum set of models but the current model where a transaction either succeeds or fails is too simple. Failed transactions should be resumable after some reconciliation of data, and computations should be notified of concurrent updates to possibly address conflicts earlier than at the end of long transactions.

Version control should be provided at the database level for provenance management and concurrent computations in workflows. Both Oracle and IBM have recently introduced temporal capabilities that keep track of the complete history of tables, providing support for some of our needs.

Asynchronous operations are required to analyze large data sets while providing continuous feedback to users. This may require trading query quality for speed as described in [14]. Additionally, during data exploration, *most of the queries submitted are extensions or restrictions of recent queries*, offering potential speedups for fast approximate queries. *Mechanisms to explicitly stop, refine, extend, resume queries* would allow more user control over the system's behavior and faster initial results.

Approximate/Partial aggregate queries would be extremely valuable for VA, as described in [11]. The proposed system continuously returns partial values of average queries while the query is being run on the whole database, allowing users to quickly see the results applied to a growing sample of data until it completes. Coupled with the previous mechanism for controlling the query, this mechanism would offer interactive continuous feedback, which is essential for VA.

While database vendors already support some of the requirements we discussed, other requirements need further research. We believe that this research will enrich both the Visual Analytics and the database fields.

We note that the increasing importance of scientific data sets has caught the attention of the database community, as evidenced by the work of Howe and Maier [13]. Recently, we are starting to see more work at the interface of these areas, and we expect that in the future, database and visualization systems will be able to be much more tightly coupled.

Acknowledgments: Silva's work has been funded by the U.S. Department of Energy (DOE) Office of Biological and Environmental Research (BER), the National Science Foundation, and ExxonMobil.

References

[1] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo. VisTrails: Enabling interactive, multiple-view visualizations. In *IEEE Visualization 2005*, pages 135–142, 2005.

[2] V. Benzaken, J.-D. Fekete, P.-L. Hémery, W. Khemiri, and I. Manolescu. EdiFlow: data-intensive interactive workflows for visual analytics. In *ICDE 2011*, pages 780–791, 2011.

[3] P. A. Boncz, M. L. Kersten, and S. Manegold. Breaking the memory wall in monetdb. *Commun. ACM*, 51(12):77–85, Dec. 2008.

[4] N. Boukhelifa, F. Chevalier, and J.-D. Fekete. Real-time Aggregation of Wikipedia Data for Visual Analytics. In *VAST 2012*, pages 147–154, 2010.

[5] F. Chevalier, S. Huot, and J.-D. Fekete. WikipediaViz: Conveying Article Quality for Casual Wikipedia Readers. In *PacificVis 2010*, pages 215–222, 2010.

[6] H. Childs, E. S. Brugger, K. S. Bonnell, J. S. Meredith, M. Miller, B. J. Whitlock, and N. Max. A contract-based system for large data visualization. In *IEEE Visualization 2005*, pages 190–198, 2005.

[7] F. Chirigati and J. Freire. Towards integrating workflow and database provenance: A practical approach. In *International Provenance and Annotation Workshop (IPAW)*. Springer Verlag, 2012.

[8] M. Cox and D. Ellsworth. Managing Big Data for Scientific Visualization. In *ACM SIGGRAPH '97 Course #4, Exploring Gigabyte Datasets in Real-Time: Algorithms, Data Management, and Time-Critical Design*, 1997.

[9] J.-D. Fekete. The InfoVis Toolkit. In *InfoVis 04*, pages 167–174, 2004.

[10] J.-D. Fekete, P.-L. Hémery, T. Baudel, and J. Wood. Obvious: A meta-toolkit to encapsulate information visualization toolkits — one toolkit to bind them all. In *VAST 2011*, pages 89–98, 2011.

[11] D. Fisher, I. Popov, S. Drucker, and M. Schraefel. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *CHI '12*, pages 1673–1682, 2012.

[12] J. Heer, S. K. Card, and J. A. Landay. Prefuse: a Toolkit for Interactive Information Visualization. In *CHI '05*, pages 421–430, 2005.

[13] B. Howe and D. Maier. Algebraic manipulation of scientific datasets. In *VLDB '04*, pages 924–935. VLDB Endowment, 2004.

[14] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The researcher's guide to the data deluge: Querying a scientific database in just a few seconds. *PVLDB*, 4(12):1474–1477, 2011.

[15] Kitware. Paraview. `http://www.paraview.org`.

[16] D. Koop, E. Santos, B. Bauer, M. Troyer, J. Freire, and C. T. Silva. Bridging workflow and data provenance using strong links. In *SSDBM*, pages 397–415, 2010.

[17] Lawrence Livermore National Laboratory. VisIt: Visualize It in Parallel Visualization Application. `https://wci.llnl.gov/codes/visit` [29 March 2008].

[18] D. A. Norman. *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*. Addison Wesley, 1994.

[19] E. Santos, D. Koop, T. Maxwell, C. Doutriaux, T. Ellqvist, G. Potter, J. Freire, D. Williams, and C. Silva. Designing a provenance-based climate data analysis application. In *International Provenance and Annotation Workshop (IPAW)*. Springer Verlag, 2012.

[20] C. Silva, J. Freire, and S. P. Callahan. Provenance for visualizations: Reproducibility and beyond. *IEEE Computing in Science & Engineering*, 2007.

[21] C. Silva, Y. jen Chiang, W. Corrła, J. El-sana, and P. Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. Technical report, LLNL UCRL-JC-150434-REV-1, 2003.

[22] H. Vo, J. Comba, B. Geveci, and C. Silva. Streaming-enabled parallel data flow framework in the visualization toolkit. *Computing in Science Engineering*, 13(5):72 –83, sept.-oct. 2011.