LA-UR-00-6116

*Approved for public release;
distribution is unlimited.*

Title:

Qviz: a Framework for Querying and Visualizing Data

Author(s):

T.A. Keahey, P. McCormick, J. Ahrens, K. Keahey

Intended for:

Proceedings of SPIE Vol. 4302 - Visual Data and Exploration and Analysis VIII, January 2001

Qviz: A Framework for Querying and Visualizing Data

T. Alan Keahey Patrick M^{c}Cormick James Ahrens Katarzyna Keahey

Advanced Computing Group
Los Alamos National Laboratory
Los Alamos, NM 87545

ABSTRACT

Qviz is a lightweight, modular,and easy to use parallel system for interactive analytical query processing and visual presentation of large datasets. Qviz allows queries of arbitrary complexity to be easily constructed using a specialized scripting language. Visual presentation of the results is also easily achieved via simple scripted and interactive commands to our query-specific visualization tools. This paper describes our initial experiences with the Qviz system for querying and visualizing scientific datasets, showing how Qviz has been used in two different applications: ocean modeling and linear accelerator simulations.

Keywords: data visualization, analytical queries, parallel processing, multivariate visualization

1. INTRODUCTION

Visualization is often a key element to allowing scientists or other users to understand large and complex datasets, along with their associated systems and processes. Effective understanding of a complex dataset comes about from gaining the ability to ask and answer questions about it, to generate and verify hypotheses. In the process of generating these hypotheses, the user may wish to test for a set of conditions having a complexity that would be difficult to identify visually in a traditional visualization environment. This may become particularly apparent when dealing with multivariate datasets, where the complexity of the visual representation increases significantly with each new attribute that is being considered. There have been a large number of tools described in the visualization literature for providing visual representations that allow the scientist to answer quantitative questions about the dataset, these include color-maps, slices, isosurfaces and other techniques. Such tools can be thought of as providing *post-visualization query* capabilities, in that they allow the scientist to infer specific numeric values at points of interest through inspection of the final visual image. A complementary technique is to provide *pre-visualization queries*, which allow scientists explicitly designate some portion of the data for special treatment before any visualization actually takes place. While we see the pre and post visualization queries to be complementary, it is worth pointing out some of the inherent differences between the two methods. Post-visualization queries can be enabled through the use of intuitive, well-understood mechanisms for embedding additional information within a visual representation. The ability to decipher a color-map is universal within the scientific community. However post-visualization queries will by their very nature be tightly bound to the visual representation of a dataset, and for some instances (such as multivariate data) where the visual representation is incomplete this may pose some problems. Add to this the difficulties that may arise through occlusion or interaction artifacts, and it is clear that in some cases a post-visualization query will result in incomplete or even incorrect results being inferred by the viewer. In contrast to these issues, we note that pre-visualization queries are not inherently bound to any specific visual representation of the dataset, but can rather be expressed in any other representation (graphical, logical, etc) that we choose. This freedom of representation comes with the potential cognitive expense of a decoupling between query and visual representations.

This paper will describe our preliminary experiences with the development and application of the *Qviz* framework for interactive querying and visualization of datasets. Qviz has been designed from the ground up to provide a lightweight, parallel, modular and extensible environment for the generation and visualization of queries on scientific datasets. Our goal is to provide *both* pre and post visualization queries in a synchronized fashion. We have placed a strong emphasis on ease of use with the system, both in terms of interactivity and learning curve, so as to facilitate its use as a tool for interactive exploration of datasets by scientists. At the most basic level, we are developing Qviz to provide quantitative tools for the analysis of datasets. At a higher level of abstraction, these quantitative tools

Email: {*keahey, pat, ahrens, kate*}*@lanl.gov*

can also be manipulated to provide additional functionalities such as feature detection, comparative visualization, multivariate data capabilities, and a general focusing mechanism. In creating the Qviz framework, we are attempting to address all of these design considerations in a single, intuitive environment that promotes the "query paradigm" as a first class interaction technique for scientific datasets. In contrast to existing full-featured visualization packages however, our focus is solely on methods for expressing queries and viewing their results. The extensibility of our framework allows the inclusion of additional visualization packages where additional functionality is desired.

2. THE QVIZ FRAMEWORK

The Qviz framework consists of a set of discrete components that can be interconnected in a variety of ways. Each component corresponds to a particular stage in our query-visualization process: data specification and preprocessing, query specification, query retrieval, and visual presentation. The *Query Registry* is responsible for dynamically assembling these components according to run-time configuration scripts and user input. In the following four subsections we will briefly describe each of these components. Following that, we will discuss some of the different ways in which these components can be assembled.

2.1. Data Specification and Preprocessing

This stage involves a short script that defines any number of datasets to be read into memory. For each dataset specified, we define the type of the data (currently scalar and vector fields are fully defined for regular grids), a symbolic name and one or more filenames containing the data for that dataset. In addition, Qviz can compute elementary mathematical expressions on datasets such as addition, multiplication or normalizing. As each expression is evaluated, the result is stored as another dataset having its own symbolic reference which can in turn be referenced by later expressions. This nesting of expressions allows expressions of arbitrary length and complexity. The script fragment below shows the definition of two scalar fields for temperature and salinity, and a 2D vector field for flow. Then we derive a field for the square of the temperatures, and normalize the salinity to between 0 and 255. An optional DUMP flag can be specified for each dataset to indicate to the preprocessor that it should be saved to a file after it has been loaded and/or computed for later processing.

```
DATADIR ../../data/1280x896/
DATA SCALAR temp       T.k2
DATA SCALAR salinity   S.k2
DATA VECTOR uvflow     u.k2  v.k2
DATA SCALAR temp^2     EXPR temp * temp
DATA SCALAR salNorm    EXPR salinity || 255
```

This preprocessing facility is similar to the derived fields for CFD described in,[1] and the compute modules of IBM Data Explorer[2] (which also provides comparison operators). In contrast to the rich set of operations described in those works, our intent with this system is simply to provide elementary facilities for the types of basic data manipulation that we frequently encounter when processing data for subsequent visualizations. However, as new data operations are required it is a relatively simple matter to extend our system to provide them. We expect that as we encounter more applications, our library of data operations will grow to accommodate their needs.

2.2. Query Specification

Now that we have defined the datasets to use as inputs, the next stage involves the specification of queries to perform on those datasets. Out scripting language for specifying these queries is similar to that used in the data processing stage, although the inherent complexity of the types of queries that we want to express is much greater than the relatively simple data preprocessing operations.

A query configuration script is composed of any number of *Named Queries*. A single Named Query is composed of a tag providing a symbolic name for the query, a *Query Expression*, and an optional return type. The simple example below shows a single named query called *tempQ* which specifies finding all areas in the dataset *temp* where the value is less than or equal to 2.0. More complex queries will be shown in the detailed explanation of query expressions and return types below.

```
QUERY tempQ  VALUE temp <= 2.0   END
```

A Query Expression is composed of a type specifier, a dataset on which to perform the query, an optional negation operator, a query operator, and one or more parameters. We have currently implemented three type specifications: value types query the actual values contained in the dataset, index types extract regions of the dataset based on their index locations, and macro types provide references to the results of previously defined queries. The dataset specification is a tag referring to either a dataset defined in the data configuration script, or to a previously defined query (thus allowing queries on queries). Examples of allowable query operators are the standard comparisons ($<, >=, =$), range, isovoxel (similar to an isosurfacing operation), and vector magnitude and angle operators. Additional operators can be easily added to the framework by compiling a new method that performs the operator into the framework, and updating the parser to provide a pointer to that method when the operator is encountered. A Query Expression can either be a single query, or any number of single queries connected by logical boolean operators. Such *Accumulation Queries* are evaluated left to right, if a more complex order of operations is required then the user can use the macro facility to specify any arbitrary order of operation. While the boolean combinations provided within an Accumulation Query are somewhat limited expressively, they can offer significant performance advantages over the use of macros for many of the standard types of queries that are encountered. This will be discussed in more detail in the following section on Query Retrieval.

The following script fragments show some examples of typical queries that can be constructed. The second Named Query combines a macro reference to the first query with a value query to find all areas where the temperature is between 5.0 and 6.5, where the salinity is greater than 0.036. The final Named Query combines the preceeding three queries to find all regions where the vector angle is between 0.1 and 1.6 radians, the vector magnitude is between 80 and 90, and the indices for the region are between rows 500 and 600 and between columns 200 and 300.

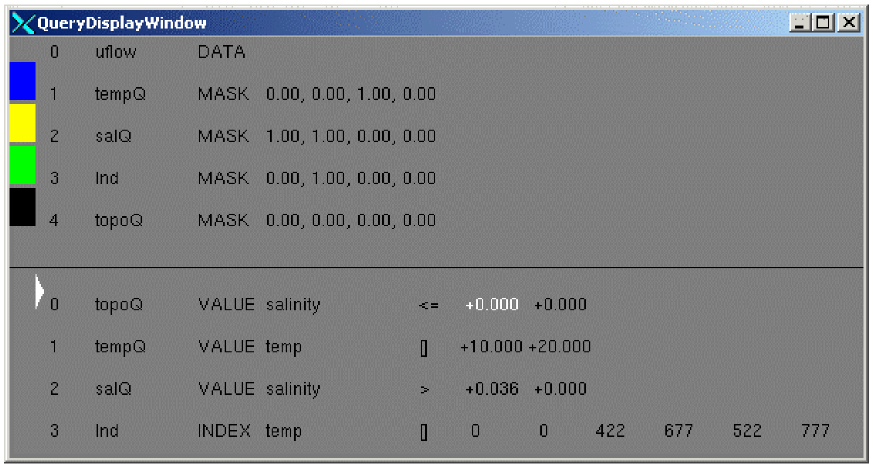
```
QUERY tempRangeQ VALUE temp       []  5.0 6.5               END
QUERY salTempQ   MACRO tempRangeQ  &  VALUE salinity > 0.036 END
QUERY uvAngQ     VALUE uvflow     //  0.1 1.6               END
QUERY uvMagQ     VALUE uvflow     ||  80.0 90.0             END
QUERY Index      INDEX temp       []  0 0 500 600 200 300   END
QUERY uvAngMagQ  MACRO uvMagQ  &  MACRO uvQ  &  MACRO Ind    END
```

For the optional return type specification, Qviz currently provides three ways in which the result of a query can be encoded: boolean, pass-through, and bitwise. The default boolean result type creates a mask containing only true or false values. In contrast, the bitwise and pass-through return types allow us to embed the query result within the existing data. The pass-through return type passes through the value that is being queried everywhere that the query condition is met, and a runtime-designated NULL value everywhere else. The bitwise return type always passes through the value being queried, after first encoding the pass/fail status of the query condition in a *query bit*. Subsequent queries in an accumulation query perform boolean operations on the specific query bit, rather than passing through the later values that are being queried. This type of bit-twiddling to encode the information can only be performed if the data is of an integral or enumerative type, and will not work with floating point data. Multiple query bits can be specified when using the query macro mechanism. The result of this return type is a slightly lower resolution dataset (in terms of range of the data values, not spatial resolution), in which the distribution of the query condition(s) can be viewed in-situ with a dataset via manipulation of color-maps and texture lookup tables.

A few simple script fragments below illustrate the syntax. As with the dataset configuration scripts, an optional DUMP keyword can be used to tell the Query Registry to save the results of a query to file after the computation is finished.

```
QUERY tempQ      VALUE temp      []  0.6  6.5  BOOLEAN  255 END
QUERY salinityQ  VALUE salinity  >   0.036     PASSTHRU   0 END
QUERY densityQ   VALUE density   <   100       BITWISE    0 END
```

In addition to the scripting interface, Qviz also provides a simple GUI interface for displaying the current set of queries stored by the Query Registry. This Query Controller also allows the user to interactively manipulate the parameters of the queries to better refine them through an iterative exploratory process.



```
QueryDisplayWindow
   0   uflow   DATA
   1   tempQ   MASK   0.00, 0.00, 1.00, 0.00
   2   salQ    MASK   1.00, 1.00, 0.00, 0.00
   3   Ind     MASK   0.00, 1.00, 0.00, 0.00
   4   topoQ   MASK   0.00, 0.00, 0.00, 0.00

   0   topoQ   VALUE  salinity   <=   +0.000  +0.000
   1   tempQ   VALUE  temp       []   +10.000 +20.000
   2   salQ    VALUE  salinity   >    +0.036  +0.000
   3   Ind     INDEX  temp       []   0       0       422     677     522     777
```

Figure 1. The Query Controller for manipulating query and visualization parameters. The bottom section of the window shows 4 different queries (labelled *topoQ, tempQ, salQ, ind*). The top section of the window shows the visual representation (type and RGBA values) to apply to the named query results. These are the queries and parameters for the 2D ocean example described in Section 3.2 and illustrated in Figure 3.

2.3. Query Retrieval Engine

After the data and query configurations are defined, an update mechanism is invoked to perform the actual queries. This update mechanism is the computational core of the query visualization process, and there are a number of efficiency and correctness issues that need to be considered in its design. The update mechanism is also invoked as the user interactively manipulates the individual parameters of specific queries. The update needs to account for dependencies created in the configuration of the data and query specifications. The use of primitive data expressions and query macros entails that it will traverse the macro dependency graph to update all queries and data expressions that are dependent on the current dataset or query and it's descendents. Since Qviz currently stores its data and query expression macros in a linked list representation, we can achieve this dependency update through a modified left to right traversal of the list.

Processing queries lends itself naturally to both data and task parallelism. So far we have introduced data parallel processing in our implementation taking direct advantage of shared memory for synchronization and data storage. Since most query processing is embarasssingly parallel, the speedup is directly proportional to the number of processors at our disposal which makes a big difference in interactivity even for relatively small number of processors. For example execution of a comparison query on our linear accelerator dataset of 512^3 cells, running on 64 processors of an SGI Origin2000, results in the near-interactive update time of 2 seconds per query. We are currently investigating ways in which this update machanism can be improved by the introduction of task parallelism and analysis of the dependency graph for dynamic scheduling.

The Accumulation Queries mentioned in the previous subsection do not have the same expressive power as we can get by combining query macros (in particular, order of operations can only be left to right), however they represent a class of queries that can be computed more efficiently than when using the more general macro mechanism. When a Named Query is defined, space is always allocated for the storage of the result, thus as complex expressions are developed the storage increases proportionally. In contrast, Accumulation Queries use a single storage area in memory into which the entire expression is computed. This not only decreases memory usage, but also increases data locality for increased efficiency. Despite their expressive limitations, Accumulation Queries are actually expressive enough to account for a great deal of the types of queries that are typically encountered.

2.4. Visual Presentation

The final stage of our query visualization process involves the visual presentation of the query results to the user. Up until this stage, all of the processing has been independent of the method that is used for displaying the results. Although the type of visualization that is used will necessarily be somewhat dependent on the types of data and queries (2D vs. 3D, vector vs. scalar, etc...), we have designed the Qviz framework to keep these as independent as possible. By keeping our query and visualization stages modular, we make it easier to plug in different visualization routines to look at the query results in different ways. While it is important that the visualization tools used should mesh effectively with the query results, we do not want to tie ourselves to any fixed set of visualization methods. Qviz provides a number of tools for visualizing query results, however it is also easy to export the query results to other visualization packages. The Qviz scripting language provides a simple and direct means for specifying how the built-in visualization tools should be used to show the results of a set of queries.

One of our major design principles in developing visualization tools for the query visualization task is that where possible we should present the query results *in-situ*, so that the user can see both the data and the query results simultaneously. Two fundamental methods by which this can be achieved are *masking* and *layering*. Both of these methods are similar in that they use a query result to alter the visual appearance of a given dataset, deciding which one to use is dependent on how you want to interpret the query results. For masking the query result is used as a boolean mask to fully clip out regions that do not meet the query condition. This can be helpful in reducing the complexity of the visualization, and allowing the user to focus on regions of interest. Similar techniques have been used for clipping geometry[3] and thresholding.[4] Layering provides a modification to the masking concept by rendering the query results as semi-transparent layers on top of the underlying data. This allows us to highlight regions of interest where the query condition has been met, or to dim or subdue the regions that do not meet the query condition. By rendering each query result as a layer with a different color, we can also use layering to show the results of multiple queries simultaneously. This layering provides a basis for an elementary type of multivariate visualization.

Another major principle for the design of the visualization tools for query visualization is to provide the visual analogy of logical combinations of data and queries. While previous sections have described facilities for logically combining query results via logical connectives, we also wish to develop methods for combining these results visually. Layering is an example of a means of visually combining queries in a way that conveys more information than could normally be obtained through simple boolean combinations of queries. By rendering different queries as translucent layers, the user can instantly detect *both* the unions and intersections of these queries simultaneously.

Qviz provides built-in tools for visualizing both 2D and 3D query results. The 2D visualization tools make extensive use of masking and layering to show the results of multiple queries. Masking in 3D is a relatively straightforward operation, however layering of query results provides a greater challenge due to issues of occlusion and transparency. Qviz provides a texture memory-based volume visualization tool for visualizing query results. With used in conjunction with the bit-wise or pass-through return types, it is possible to produce fairly sophisticated results where regions of the data matching the query (or queries) can be made more or less transparent by the user. This provides an *in-situ* representation in 3D where results of complex queries can be seen in place with the original dataset. Modifications to the transparency can be performed interactively by the user with very rapid response rates through simple manipulation of the texture lookup table. Earlier work on the general task of viewing multiple values in a volume by slicing and sub-setting includes.[5]

Although we would prefer be able to visualize our query results *in-situ* wherever possible, there will also be occasions where the complexity of the queries or dimensionality of the dataset is so high as to require alternative visualization methods. Towards that end, Qviz allows multiple views of datasets and query results. Any number of windows can be created, each having its own view of particular attributes of the data or query results.

The script fragment below shows how a few different visualization effects can be specified, in general a viz specification script consists of any number of viz specifications, each composed of: the VIZ keyword, the symbolic name of a dataset or query results, the desired visual representation, and optional parameters specifying the RGBA components to use in the visual representation. The order in which the lines are scripted determines the order of layering (if any) for the visual representation of the results.

```
VIZ uflow    DATA
VIZ topoQ    MASK    0.0 0.0 0.0 1.0
VIZ salQ     MASK    0.8 0.1 0.8 0.5
VIZ tempQ    MASK    0.8 0.1 0.1 0.5
```

In the script above, the first line indicates that the dataset “uflow” should be rendered using the default or current colormap. The second line indicates that the results of the “topoQ” query should be rendered as an opaque back mask on top of the “uflow” data, the third and fourth lines specify that the salQ and tempQ query results should be rendered as semi-transparent layers on top of the previous layers. In addition to the DATA and MASK keywords, the BITWISE and PASSTHRU keywords can be used to indicate to the visualization tool that these special types of rendering should be performed.

2.5. Configuring the Query Registry

There are a number of useful ways in which the components of the Qviz framework can be assembled. At the heart of this configurability is a formal grammar which describes different configurations and states of the framework. This grammar defines a scripting language that is used to: specify data files and operations for creating derived fields, specify queries (inputs, types, parameters and results), specify visualization tools and parameters, and allow flexible run-time configuration incorporating some or all of the components. Additionally, the grammar provides a formal reference model on which to base implementation, and also provides a full description of the data and query computation on which to perform dependency analysis, optimizations, and parallelization.

The use of the Qviz components can be specialized in two ways: programmers can compile their own custom configurations of the framework, or users can create configuration scripts at run-time for configuring and manipulating different combinations of precompiled component functionality. The parameters can also be manipulated interactively by the user at run time. At any point during this manipulation, the current state of the framework can be saved to a script file for later use. Our standard configuration of Qviz makes use of all the stages of the framework, several other scripted configurations provide useful results however. The data field and derived field defininitions can be used as a standalone data processor for flexible batch manipulation of data files. Although somewhat pedestrian in nature, this facility actually is quite valuable, as typically a great deal of time is spent “prepping” data before we can use it in our visualization tools and much of that preparation can now be automated. The query engine can be hooked up to the data specification and output the results in batch mode for further analysis and/or visualization by other tools. Finally the visualization tools can be hooked up directly to the data specifications to provide a “query-less visualization tool. Thus while interactive query visualization of large datasets was our primary motivation in the construction of this framework, we have also found many other ways in which the framework can be used.

3. APPLICATIONS

3.1. Linear Accelerator Simulation

Particle accelerators play an increasingly important role in basic and applied science. A main goal in the computer simulation of particle beam dynamics is to understand the beam’s evolution as it propagates along the accelerator. A key feature within the beam is the halo which is responsible for beam loss (radioactivity) when particles strike the beam pipe. Understanding the behavior of the halo is a major issue for next-generation, high-current, accelerators.

High-resolution particle beam simulations use hundreds of millions to billions of particles to model the details of a beam. These particles may be mapped into a density volume as described in.[6] Figure 2 shows several images of a 512^3 density volume from a phase space representation of the beam. The phase space representation can be described as a combination of the particle locations and momenta, and is described more fully in.[6] The sequence of images shows a series of increasingly refined density queries for removing the denser core of the beam, until only the diffuse halo region with very small density remains. In each image, we have used a spatial query to remove one end of the “shell” formed by the beam, thus allowing us an unobstructed view into the interior.

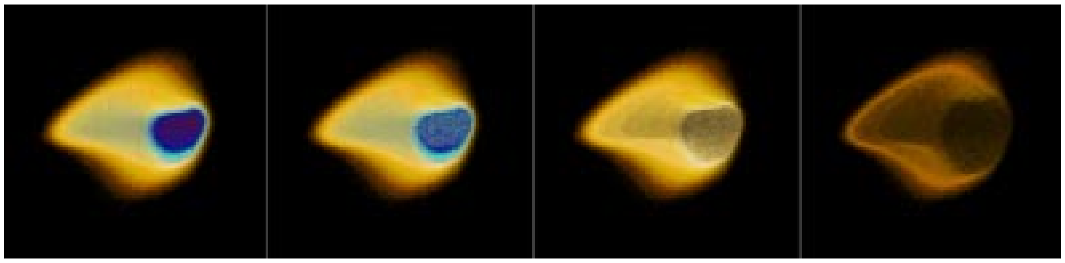


Figure 2. Phase space representation of a linear accelerator beam. The sequence illustrates removing increasing amounts of material from the core of the beam via density queries until only the outer halo remains.

3.2. Finding Features in Ocean Models

This section examines the output from a $992 \times 1280 \times 128$ cell ocean simulation of the North Atlantic. There are many regions of interest in this dataset, however we will focus primarily on the area where the warm water of the Mediterranean Sea mixes with the colder waters of the Atlantic. Ocean modelers are particularly interested in examining how their model behaves in the region where the relatively warm, salty sea water spills into the lower ocean regions. There are 5 primary properties that are being modeled in this simulation: temperature, salinity, and flow velocity in u, v and w. An additional field contains information about the topography of the ocean floor.

We extracted the top horizontal slice from the ocean model and performed some simple queries on it to obtain the result shown in Figure 3, which has the following features: a grey-scale color mapping is used to render the magnitude of flow velocity, an opaque black overlay is used to mask out the null regions in the model that are above water, a transparent blue overlay is used to indicate where the temperature is between 10 and 20 degrees (C), and a transparent yellow overlay is used to indicate where salinity is greater than 0.036. The intersection of these two regions will thus have magenta shading. From the patterns of temperature and salinity that we observe, we spatially select a rectangular region of interest (shown as a green overlay). We will extract the 3D volume associated with this region of interest for further study.

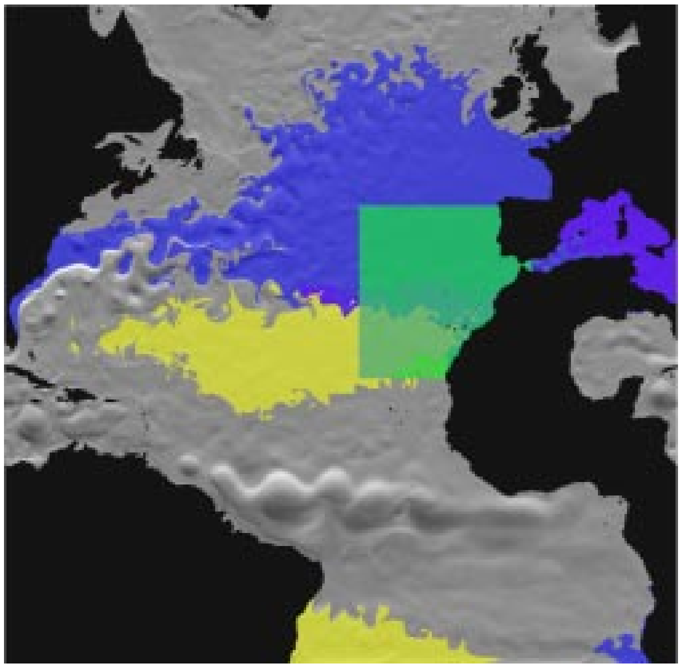


Figure 3. Top level view of a North Atlantic ocean simulation. Flow velocity is shown in grey-scale. The results of temperature and salinity queries are shown as blue and yellow overlays respectively. The green rectangular overlay marks the region selected for further study. (The grey-scale section in Africa is space "borrowed" from that null land region to represent the Gulf of Mexico.)

This example shows in 2D how multiple query results can be visually combined to show where the query conditions have been satisfied. Qviz allows the user to easily define and modify the way in which these layering operations take place. By changing a couple of lines in the script we can modify the above query to instead provide a grey-scale rendering of the temperature field with a transparent overlay showing where the flow velocity is within a certain range. Any number of these queries can be defined and shown as layers, and the user can interactively modify the parameters of the query as well as the opacity of the individual layers.

Figure 4 shows two views of the extracted data rendered with a texture-based volume renderer. In the first view we have further refined the temperature query to < 15 (C). In that view we can see that the temperature/salinity queries are satisfied in two primary ocean regions, near the surface, and down at a lower depth. In addition there is a narrow stream spilling out of the Mediterranean into the lower of the two ocean regions. However, our view of that flow is somewhat occluded by the upper region, to correct this a spatial depth query was used to remove the top region, with the result shown in the second image of Figure 4. The upper and lower regions matching the query are independent of each other; the top region is the result of warming and evaporation by the sun, and the lower region is the result of water transport from the Mediterranean.

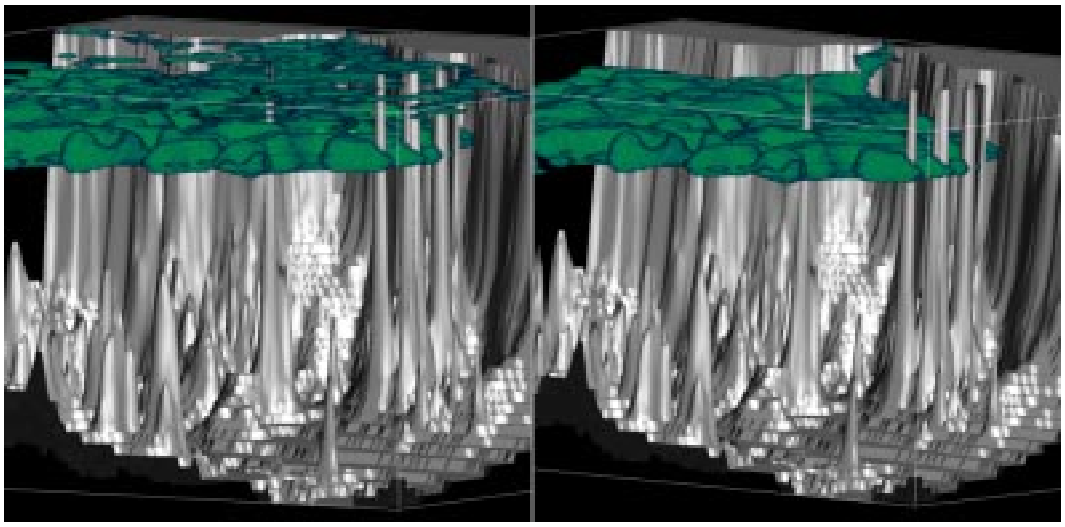


Figure 4. Volume rendered views showing the result of a temperature/salinity query to extract the body of water spilling out of the mouth of the Mediterranean. In the first image the lower body of interest is occluded by a higher ocean region also matching the query, in the second image we use a spatial query to remove the upper layer to get a clear view of the lower one.

Figure 5 shows a more detailed view of the spill region at the mouth of the Mediterranean. The query results have been exported to the Visualization Tookit (VTK[7]) where vector glyphs are used to show the flow. A query based on temperature, salinity and vector magnitude was used to isolate the region of interest. The image shows the spill of water out of the sea into deeper layers of the ocean, as well as an upper return current flowing back into the Mediterranean.

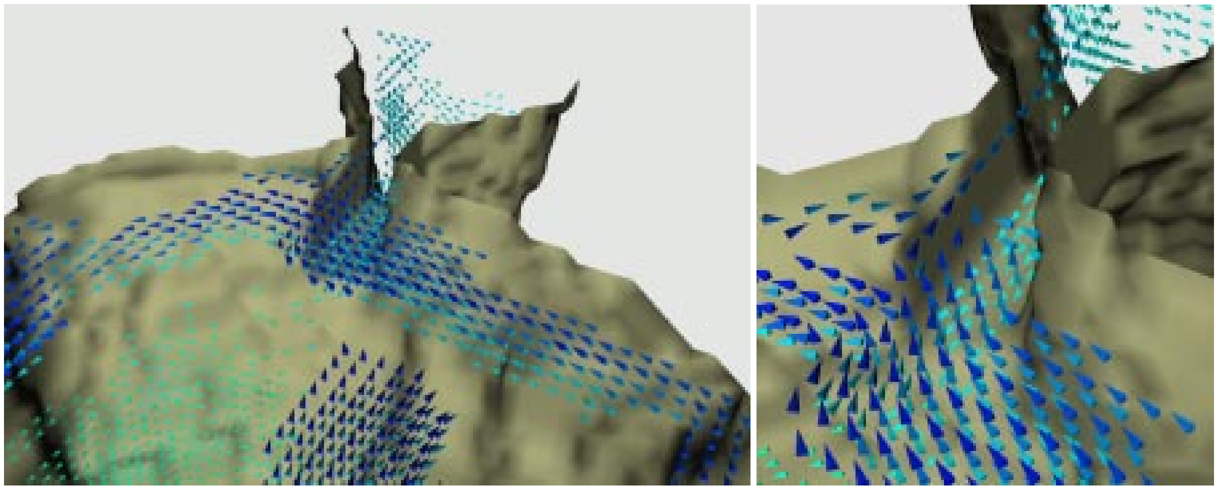


Figure 5. Detailed view of the vector flow field around the Straits of Gibraltar, filtered by a temperature, salinity and flow magnitude query. The dark blue arrows in the upper layer show water flowing into the Mediterranean, while the light blue arrows at a lower level show the salty water spilling out of the sea into a deeper pool. The view on the right provides a closer view of the specific flow vectors in the channel.

3.3. Other Applications

The preceding examples have illustrated several different types of query results that can be derived from Qviz. There are many more ways in which Qviz has been, and will be, used to glean insights from datasets. One particular area where the query facilities of Qviz will prove useful is in comparative visualization,[8] particularly in comparing the results obtained from different simulation models, or for comparing a simulation model to observational data. Our preprocessing facility allows differences between datasets to be easily computed, and the visual layering of results provides a simple but effective mechanism for relating those differences to the underlying models. Another area in which Qviz can be directly applied is as an analyzer for time series datasets, computing the difference between time steps to determine regions of variability. Qviz can also be used as a multivariate visualization tool, any number of views can be created to simultaneously show any number of variables, in a scatter-plot-type fashion.[9] We look forward to discovering further ways in which the analytical and visualization tools of Qviz can be used to extract meaningful information from scientific and other datasets.

4. CONCLUSIONS AND FURTHER WORK

Qviz is a lightweight package that offers significant advantages in terms of ease of use, data-parallelism for interactive response rates, modularity, extensibility and power of expressiveness. We have demonstrated a number of real-world applications to which Qviz has been applied. These examples illustrate some of the ways in which Qviz can be used to extract and present interesting aspects of scientific datasets, both as a stand-alone tool and in conjunction with other visualization packages. We have enjoyed working with the application scientists as they used Qviz to isolate features of the data, and we have received very positive feedback from them as they gain new insights into their data.

Our experiments with data parallelism are encouraging, we hope however that by integrating task and data parallelism and introducing macro dependency analysis, we will be able to leverage cache reuse to a much higher extent and thereby achieve more significant speedups. We are also working towards a more complete and formal description of the query visualization process. We will be integrating Qviz with our future high-performance volume rendering system to provide better presentation of large 3D queries, in addition our simple GUI for manipulating queries could use some improvement.

5. ACKNOWLEDGEMENTS

We would like to thank the members of the Advanced Computing Group's ocean modelling team at Los Alamos National Lab for providing for providing data and feedback on this system, particularly Matthew Maltrud and Bob Malone. We would also like to thank Ji Qiang and Robert Ryne of the Accelerator Physics group at Los Alamos for their collaboration on the particle accelerator application.

REFERENCES

1. C. Henze, "Feature detection in linked derived spaces," in *Proceedings of IEEE Visualization*, 1998.
2. G. Abram and L. Treinish, "An extended data-flow architecture for data analysis and visualization," in *Proceedings of IEEE Visualization*, 1995.
3. W. E. Lorensen, "Geometric clipping using boolean textures," in *Proceedings of IEEE Visualization*, 1993.
4. A. Pang and N. Alper, "Mix & match: A construction kit for visualization," in *Proceedings of IEEE Visualization*, 1994.
5. T. A. Foley and D. A. Lane, "Multi-valued volumetric visualization," in *Proceedings of IEEE Visualization*, 1991.
6. P. S. McCormick, J. Qiang, and R. D. Ryne, "Visualizing high-resolution accelerator physics," *IEEE Computer Graphics and Applications* , Sept. 1999.
7. W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, Prentice Hall, 1997.
8. A. Pang and A. Freeman, "Methods for comparing 3D surface attributes," in *SPIE Visual Data Exploration and Analysis*, 1996.
9. J. LeBlanc, M. O. Ward, and N. Wittels, "Exploring n-dimensional databases," in *Proceedings of IEEE Visualization*, 1990.