Visualization Exploration and Encapsulation via a Spreadsheet-like Interface

T.J. Jankun-Kelly and Kwan-Liu Ma
Computer Science Department,
University of California, Davis

Abstract—
Exploring complex, very large data sets requires interfaces to present and navigate through the visualization of the data. Two types of audience benefit from such coherent organization and representation: first, the user of the visualization system can examine and evaluate their data more efficiently; second, collaborators or reviewers can quickly understand and extend the visualization. The needs of these two groups are addressed by the spreadsheet-like interface described here. The interface represents a 2-dimensional window into a multi-dimensional visualization parameter space. Data is explored by navigating this space via the interface. The visualization space is presented to the user in a manner that clearly identifies which parameters correspond to which visualized result. Operations defined on this space can be applied which generate new parameters or results. Combined with a general purpose interpreter, these functions can be utilized to quickly extract desired results. Finally, by encapsulating the visualization process, redundant exploration is eliminated and collaboration is facilitated. The efficacy of this novel interface is demonstrated through examples using a variety of data sets in different domains.

Keywords—**spreadsheets, user interfaces, knowledge representation, scientific visualization, visualization systems, collaboration**

I. Introduction

In order to gain insight from large, scientific data sets via visualization, both efficient algorithms and intuitive user interfaces (UIs) are needed. Research in visualization has focused upon the former, developing techniques to generate realistic, informative visualizations quickly and economically. As these methods proliferate, powerful and informative user interfaces to make use of them become more important. By presenting and storing the visualization exploration process, this process becomes streamlined: past work can be built upon—avoiding potentially costly repetition—and results can be easily shared and reused. Towards this end, a spreadsheet-like interface that satisfies these requirements has been developed. This paper discusses its capabilities and illustrates its uses through a series of examples.

Conventional spreadsheets have three properties [1]: tabular layout, operators, and cell dependency management. Tabular organization allows quick comparison of results and structures the subsequent analysis. Cell operators can assist in this analysis by providing a suite of functions to manipulate the cell values. Cell dependency is used to allow changes in one area of the spreadsheet to propagate to others. The usefulness of these characteristics is attested by the number of different applications of spreadsheets that exist. Our spreadsheet-like interface uses similar properties to organize and control the visualization process. The tabular display acts as a window into a multi-dimensional visualization space that a user manipulates to discover results of interest. Operators on the cells can analyze and generate visualizations. Similarly, parameter operators can be used to further exploration. As cells represent fixed points in visualization space, traditional spreadsheet cell dependency management does not apply. This is offset by an interpreter that can modify the visualization at a lower level. The interpreter allows experts to perform complex operations upon the data that supersede the UI's facilities. Our interface builds upon the strengths of spreadsheets while augmenting them towards the task of visualization exploration.

Besides presenting an interface to the visualization process, our spreadsheet also captures that process for the user. This encapsulation of the history is crucial. As the size of scientific data increases, users of visualization systems must be able to explore this data efficiently. Redundant exploration, a potentially expensive operation, is avoided by storing and displaying previous results. In addition, the user gains a clear picture of what has and has not been tried. This information can then be shared with others to communicate both the results and the steps used to generate those results. These details are discussed in the following sections.

II. Visual Representations of Data Exploration

The reuse of visualizations is an important issue, especially when generation of the visualization is costly. Results from a previous visualization can suggest parameters for later investigation. A user interface which exposes these previous results will make subsequent exploration of the data easier. The interface also needs to transparently allow the user to navigate through and manipulate the underlying data. If the interface is cumbersome in either its usage or presentation of information, the user cannot properly analyze their data. The user interface must both intuitively display the progress of the visualization session and allow visual manipulation of this progress. Such interfaces are *visual representations* of the visualization process. An effective visual representation of the data exploration should

Visualization and Graphics Research Group, Center for Image Processing and Integrated Computing, Computer Science Department, University of California, Davis, CA 95616. E-mail: {kelly, ma}@cs.ucdavis.edu

meet the following criteria:

- *Parameter Manipulation*: The user must be able to set and manipulate parameter values to generate visualizations.
- *Intuitive Display*: The relationship between and context of different visualizations must be displayed.
- *Visualization Operators*: Parameter and value operators to extract information and generate new visualizations must be provided.
- *Process Encapsulation*: The history of the exploration process must be captured for later collaboration.

These properties build upon each other. For example, the history of the visualization session should utilize the intuitive display used for the main interface; this would lessen the conceptual distance between the results and the steps used to generate the results. [2]. Our spreadsheet-like interface has been designed to satisfy these criteria.

A. Turn-Key

In traditional turn-key visualization user interfaces, a user iteratively changes parameter values directly in order to search for the desired result. This trial and error process is inefficient and does not communicate context that directs a user toward their goal. Automatic systems that generate parameter values can help this process, but their result is lost if the user subsequently modifies a parameter value. Once an acceptable visualization result is obtained, only the final parameter settings and image are available to be recorded and shared with collaborators; all previous results are lost. While perhaps sufficient for prototypes, these user interfaces do not allow sophisticated control of the exploration.

B. Data-flow

Data-flow interfaces represent the data exploration process by a directed network of connected components. These components act upon the data sets or output of other states to produce their final result(s). Each component in the network represents an operation or transformation on the output of the previous step. Components can set parameter values for subsequent visualization techniques or perform other tasks such as annotation or image cropping. Several commercial systems use a data-flow interface for visualization [5], [3], [4].

Data-flow interfaces have a better state display than traditional turn-key interface through the data-flow graph: the graph clearly displays the flow of information whereas a turn-key interface has no such contextual information beyond the current state. This flow graph can be shared with collaborators to communicate the process needed to generate the final result. One weakness of the data-flow approach is that it does not indicate the history of the visualization process. Like the turn-key interface, changes to the settings of one of the flow nodes cause the previous image to be lost. After several iterations, if the user wishes to revisit a previous collection of settings, there is no obvious assistance from the user interface to support this task.

C. Parameter-based Representations

Unlike representations which focus on the flow of data through a system, parameter-based representations focus on the changes of parameter values during the visualization process. These user interfaces manipulate the operands of the visualization techniques (the parameters) directly whereas data-flow system encapsulate the parameters within the techniques. Two interfaces of this type are the Design Galleries system [6] and image graphs [7].

The Design Galleries system considers data exploration a process of exploring a multidimensional space of visualization parameters. The results a user desires exist within this space. It is the system's job to aid in the discovery of the parameters that correspond to the images. After a pre-processing rendering stage, the system displays a 3D representation of the design space. A user then navigates this space to find their desired images. By replacing a trial and error approach with a structured navigation of parameter values, the system allows a more efficient exploration.

The image graph system follows a similar structured approach. Unlike the Design Galleries system, image graphs are built dynamically instead of during a pre-processing stage. An image graph is a graph representation of the visualization process that distinctly displays the relationship between generated images via glyph edges. The graph is used to explore the space of visualization parameters. As more visualizations are added, the graph structures itself so that related images are clustered together. A user can manipulate this structure as they desire. Operations upon the edges and nodes in the graph can be used to generate further results. The graph can be shared, thus providing a history of the final result with the result itself.

Unlike the previously mentioned user interfaces, both of the interfaces display the history of the data exploration process. However, these interfaces posses two key limitations. First, screen space becomes limited as the number of different parameter settings increases; though screen real estate management is a concern for all visualization user interfaces, the rate of growth of parameter settings makes this issue paramount for parameter-based interfaces. As more images are added, zoom techniques, graph compression, or focus+context techniques must be used in order to display the entire graph [8]. Thus, comparisons between results become more difficult as more images are added. Second, both interfaces are limited by their display and manipulation of a single data set at a time. This prevents cross data set comparisons or operations.

D. Spreadsheet Representations

Spreadsheets are a subset of form visual programming languages where the user programs the contents of cells aligned in a grid [9]. Spreadsheets are most familiar from numeric applications. The visual language community has extensively studied spreadsheets and their applications. For example, the Forms/3 system [10] has been used to discuss spreadsheet animation, dynamically expanding grids, and psychological factors in designing user interfaces [12], [11], [2].

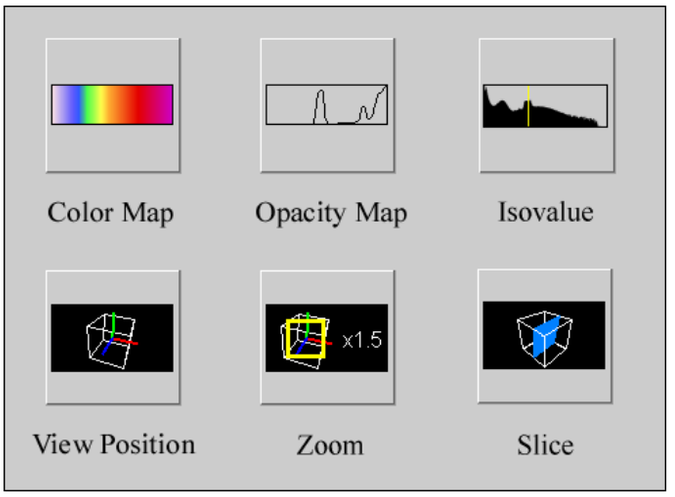


Fig. 1. Visual representation of some parameters displayed by the visualization spreadsheet. As a user edits the underlying parameter, the icon of the parameter is updated.

Several spreadsheet interfaces for graphics and visualization have been developed. Levoy's SI system [13] wraps a spreadsheet around a general image processing kernel; each cell represents a script which can reference other cells to generate subsequent images. Hasler and Palaniappan have experimented with a series of spreadsheet-based interfaces to represent satellite and other earth-observatory equipment data [14], [15]. Chi, et al. [1] demonstrate a set of principles for visualization spreadsheets through their SIV system. All of these systems are characterized by their tabular display, cell operators, and management of cell dependencies as discussed earlier.

There are several unresolved issues when using these previous spreadsheets as an interface for visualization exploration. Though they posses a structured environment for display—and thus mitigate some of the display issues of the parameter-based representations—they do not supply such structure for the actual exploration. These interfaces also "collapse" the entire multi-dimensional visualization parameter space onto a two dimensional window. Thus, there is no mental metaphor to assist the user in understanding and navigating the visualization space. These issues were addressed by our spreadsheet-like interface described in [16]. This work was subsequently expanded by formalizing the model, applying it to different visualization domains, and developing an off-line format to capture the visualization process. With these additional properties, our spreadsheet system becomes a robust solution to the parameter exploration problem for visualization.

III. Spreadsheet-based Visualization Representation

Before we discuss the structure of our spreadsheet-like interface, we must first discuss the conceptual model behind it. As suggested in Spence [17], the purpose of this conceptual model is "to have a better understanding of the artefact, scheme or situation to which the data refers, and to be able to interpret the model in some useful way." The formalism behind our internal model describes the properties of the interface.

A. Conceptual Model

Like the Design Galleries and image graph systems, we consider visualization exploration a process of examining a multi-dimensional space V of parameter values. Each n-tuple $p \in V$ represents a combination of parameters that produce a visualization. A tuple p uniquely identifies a point in the visualization space which corresponds to a particular visualization result r in some result space I. V thus defines a transformation $v : V \mapsto I$ from the parameters to the results. This transformation is the underlying visualization technique used. This definition is independent of the actual application of the spreadsheet. For example, different visualization parameter spaces exist for direct volume rendering, isosurface extraction, and vector visualization. Parameters such as color or opacity maps would be used in the first case, while stream line seed location and ejection rate would be used in the last. The representation of the visualization result could vary over domain as well: in direct volume rendering, results are rendered images; in isosurface visualization, the extracted triangulated surface.

The key feature of this approach is that it decomposes the visualization process into a sequence of parameter settings that generate results. As the user iteratively changes parameter values, they trace out a path $P = \{p_0, \ldots, p_n\}$ through the visualization space where p_0 is the first tuple of parameter settings explored and p_n is the last. This path can be viewed using the interface, manipulated with the parameters discussed later, and shared with others. Only the domain of the visualization process stored (represented by a particular v), the path P, and its corresponding results $R = \{v(p_0), \ldots, v(p_n)\}$ are needed to recreate a visualization session. The results R are not strictly needed to recreate a session, though it is often beneficial to maintain them as generating R can be costly (especially for very large data sets). The results are also needed if they will be shared outside of the spreadsheet environment.

B. Display and Navigation

A spreadsheet presents a tabular view of its underlying data. In numerical applications, this is a 2D array and thus the correspondence between data and display is trivial. Visualization space is higher-dimensional and therefore more complicated to display. Our spreadsheet is a movable, scalable window into this space. By manipulating the visualization parameters, the user changes the position and size of this window. Unlike previous spreadsheet designs, this spreadsheet places constraints upon the cell values. The spreadsheet displays a planar projection spanned by two axes of the visualization space. Only a single type of parameter value can be displayed in the rows and columns. Using volume visualization as an example, the rows could display color maps while the columns show opacity maps. For the other, non-displayed parameters, a set of default values is maintained. These values may be updated at run-

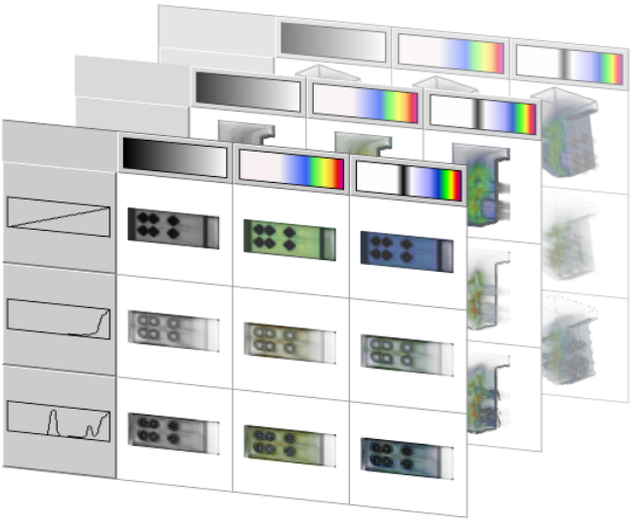


Fig. 2. Our spreadsheet is a view of two dimensions of a visualization space. In this example, opacity maps are displayed along rows and color maps along columns. A particular cell is rendered by combining the non-displayed parameters' default values with the parameter values corresponding to the row and column indices. By changing the default parameters, in this case the view position, the spreadsheet's position in visualization space can be moved.

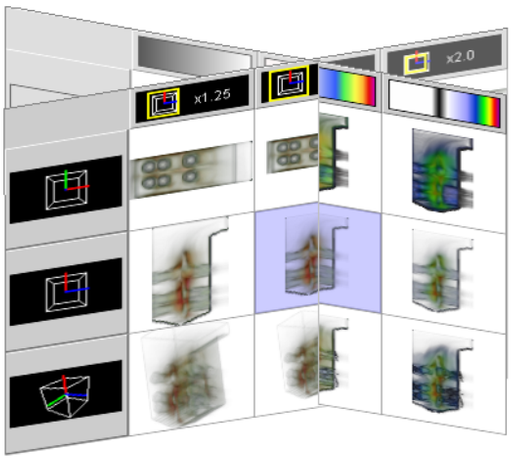


Fig. 3. The spreadsheet window can also be rotated in visualization space. Starting from a sheet displaying color and opacity maps, the user first selects an image with the desired properties. These two parameters will become the new default values. By then selecting two new parameters to display, in this case view position and zoom factor, the window is rotated about the selected point to display the new values.

time. Parameter values are represented by rendered glyphs in the table headers (Figure 1). A cell in the spreadsheet represents a result which combines the parameter values of the row and column intersecting the cell with the default values for the other parameters. By changing the the default values for non-displayed parameters, the spreadsheet "window" can be translated in visualization space (Figure 2). A different kind of motion is achieved by changing the displayed row and column parameters (Figure 3). Thus, the data exploration process becomes the process of manipulating the spreadsheet window through visualization space.

Previous visualization spreadsheets collapse visualization space into 2D without controlling what values are used in the spreadsheet cells. While this may be useful to display final visualization results side by side, this projection hinders exploration efforts since the relationship between parameter values and result is not immediately evident.

IV. Static Spreadsheet-based Exploration

Without the dynamic operators and the interpreter described later, our interface can still be used to explore a user's data. In this "static" mode, the user manipulates the spreadsheet's position in visualization space and selects which parameters (and thus results) to investigate. Actual exploration of the data would generally combine both phases: the data is explored using the static interface to generate a few images and then the dynamic operations are used to create further results.

When starting a new visualization session, a user must first initialize any parameter values that do not have default values. Using the spreadsheet, the user can select which parameters to display along the rows and columns, and add, edit, remove, and position column and row values as desired. Depending on the application, all cells or only those specifically requested by the user may be displayed—the overhead of rendering the entire table should determine which method is used. If the selected row or column parameter is changed, the table is populated with images corresponding to the new combination of parameters. If one of the non-displayed default parameter values is changed, the images are updated as well. The system visually identifies which parameters correspond to an image by rendering the row and column labels as glyphs.

Figure 4 demonstrates a spreadsheet-driven visualization. The user wished to display separate skin and bone surfaces for a foot medical data set using a ray-casting direct volume renderer. First, the user added two opacity maps which highlight the desired surfaces; the tabular organization of the spreadsheet allows the two images to be easily compared side-by-side. After changing the row parameter to display view positions, the user selected a view to display the front of the foot. This position was selected as the new default. Afterwards, the user returned to modifying color maps. Only images utilizing the new view position were displayed. Two new color maps were added, the first a false-color map highlighting differences in value on the surface and the second a color map to display a flesh-like tone for the skin and white for the bone. The latter color map was selected as the new default color map. Finally, the final images were generated by displaying and adding a new zoom factor value. If the user wanted, they could change the default color map or view position to examine alternate zoomed images.

Tabular organization is one of the advantages the spread-

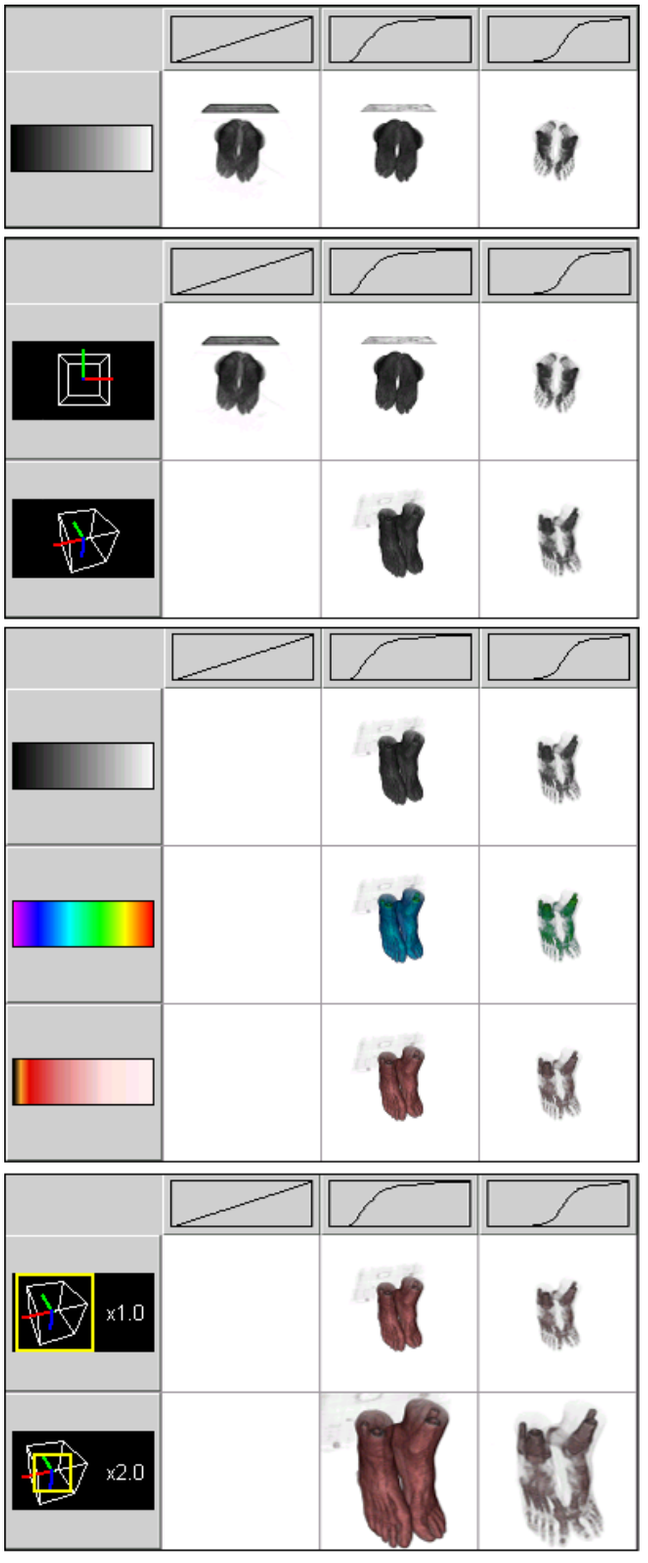


Fig. 4. A sequence of spreadsheets displaying the volume visualization of a foot data set. Blank cells represent non-rendered images. The goal was to compare skin and bone surfaces. The user first determined appropriate opacity maps before modifying the view position, color map and zoom factor. The spreadsheet was useful in displaying the images to be compared side-by-side.

sheet has over other representations. As demonstrated above, it allows quick visual comparison of data values. This property is especially useful in comparing renderings of different data sets. Figure 5 displays an example sequence of data sets representing time steps in a material propagation simulation. Changing or adding a parameter value in the figure would affect all the data sets at once. The equivalent task would require several separate operations in an image graph. The tabular structure also suggests natural parallelism when applying the operations from the next section: new cells generated by an operation could be distributed to separate processors to be visualized.

V. Dynamic Spreadsheet-based Exploration

The spreadsheet display can be dynamically modified to supplement the data exploration experience. The dynamic capabilities include both parameter and value operators and the associated interpreter. Animations can also be created.

A. Parameter and Value Operators

Like the numeric functions in traditional spreadsheets, most spreadsheets define a set of operations that can be applied to the cells. In visualization spreadsheets, operations allow the user to create new results from previous ones. Our spreadsheet defines two types of operations: those acting on parameters (row and columns) and those acting on values (cells). The former typically generates new parameters from their input while the later analyzes cell values.

There is a set of operators for each parameter type: set operations can be applied to color and opacity maps, histograms can be derived from data sets, or isovalues can be interpolated. To apply a parameter operator, a user first selects a range of column or row values to be used as the operator's arguments. The user then selects an operator to apply and, if necessary, customizes its behavior. The first spreadsheet in Figure 6 displays the illuminated magnetic field lines in a Tokamak simulation [18]. The purpose of the visualization was to discover "magnetic islands" inside the toroid object: i.e., magnetic field surfaces that do not form closed toroids. From the first spreadsheet, it is not immediately clear whether the first and last field line (measured in increasing radius from the center of the torus) enclose the second. When a union operator is applied to the parameters, combining the display of all three field lines into a single image in the second spreadsheet, the artifact is clear.

Value operators are applied in similar manner to parameter operators: the operand cells are selected and a operator is chosen from a list of possible operations. What is unique about value operations is how the cells are selected. As the spreadsheet data is multi-dimensional, cells from alternate "stacks" of the spreadsheet (i.e., different projections of the visualization space) can be selected at the same time. If a user wanted to combine the opacity and color maps from one image with the view position and zoom factor of two other images in a volume rendering example (Figure 7), the user could follow these steps:

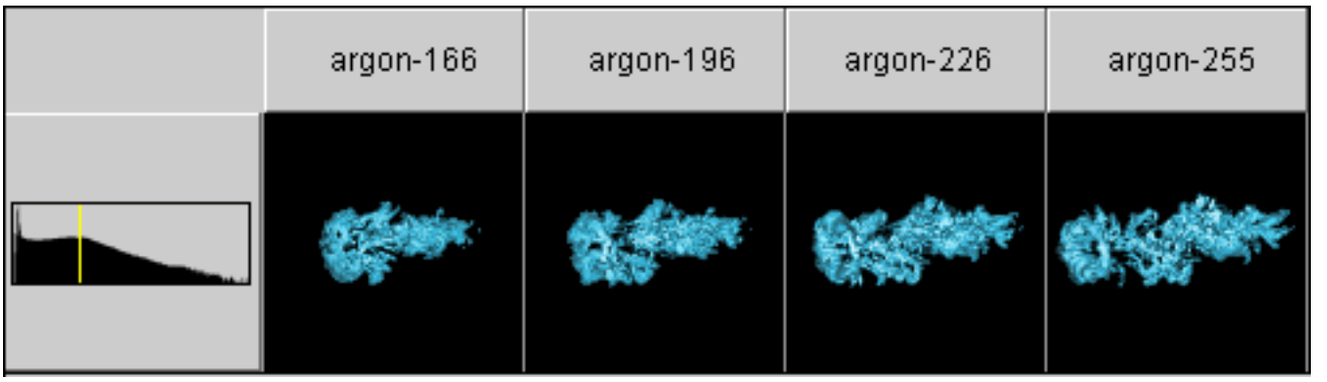


Fig. 5. An isosurface visualization spreadsheet displaying the effects of a shock wave on a bubble of argon over several time steps. Modifying the displayed isovalue would change all the cells at once, a task that would be more difficult in other representations.

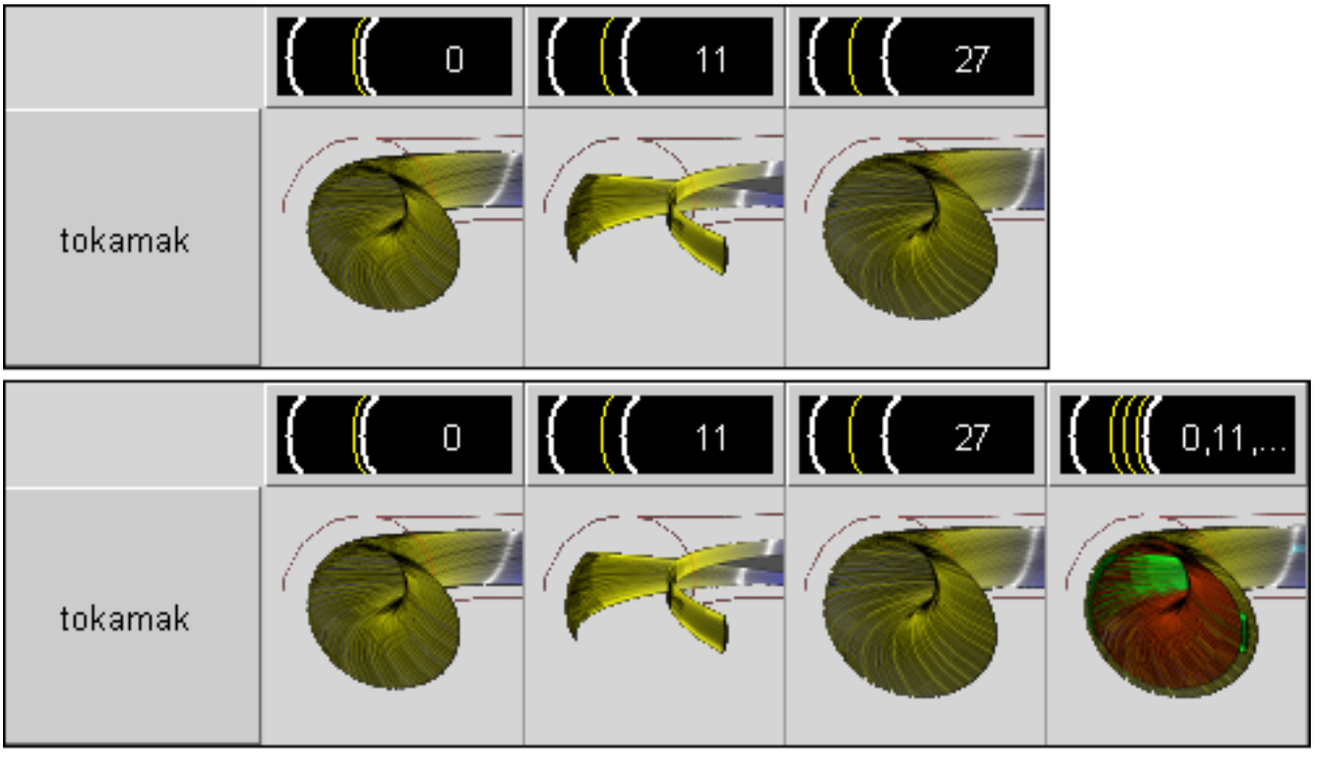


Fig. 6. An example of applying parameter operators, in this case the union of displayed magnetic field lines in a simulation. The top image displays the three different field lines side by side. When combined, the presence of the "magnetic island" between the layers becomes apparent.

1. Change the row and column parameters to display color and opacity maps.
2. Select the cell with the desired color and opacity maps.
3. Change one of the parameters to display view positions.
4. Select the cell with the desired view position.
5. Change one of the parameters to display zoom factor values.
6. Select the final cell with the desired zoom.
7. Apply the combination operator.

The new cell would then be added to the spreadsheet at the intersection of the four selected parameter values. Without cell selection in separate stacks, value operators could only be applied within a given display, limiting changes in parameter to the current row or column parameter only.

B. Animation

Unlike static images, animations better display 3-dimensional features of data. Animations are generated using the same method used to apply value operators. First, a range of key frame cells, most often from different stacks, is selected in the spreadsheet. As more than one parameter can change between images, the order of interpolation is then selected by the user. Finally, the user determines how

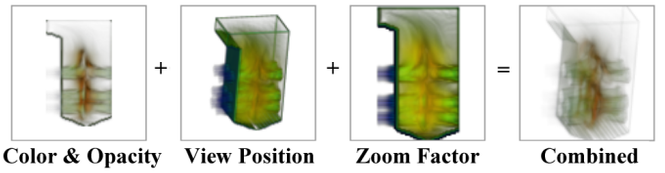


Fig. 7. Selected cells for a composition operator. The user wished to use the color and opacity maps of the first image, the view position of the second and the zoom factor of the third. The fourth image displays the desired visualization.

many intermediate steps to render between each key cell. The system then automatically renders the animation.

As an example, consider an animation using the first and last cells in Figure 7. Between these two images, both the view position and zoom factor has changed. To generate the animation, the cells containing the two key frame images would be selected. Next, the user would determine the order of interpolation: view position followed by zoom factor or vice versa. The corresponding animation would illustrate the 3-dimensional features of the flow through the furnace.

C. Scripting

Another dynamic capability of the spreadsheet is the use of an interpreter. This interactive environment allows a user to directly access and manipulate the underlying conceptual model the interface presents. The parameter and results can be used in user defined functions to programmatically generate new visualizations. All of the operators mentioned in the last sections can be utilized by the user. The usefulness of such an environment has already been demonstrated [19]. The interpreter grants advanced users low-level control of the visualization process that the UI abstracts. For example, the script

```
for i in range(3):
  addParameter( "View",
                View( xangle=45*i,
                      yangle=-45*i,
                      zangle=0 ) )
```

would generate a series of view positions in an arc about the data set. This technique can be extended to generate a series of parameters within the parameter space similar to the method used by the Design Galleries system. The user can then use the generated results to narrow their search.

Our implementation of scripting differs from common macro languages in numerical spreadsheets or the scripting abilities in the spreadsheets described by Chi et al. [1]. In these applications, cells are referenced by their row and column values. If a cell's value changes, all formulas which reference that cell are updated; these changes are propagated as needed. In our spreadsheet, the cells represent immutable points in space. Similarly, the parameter a row or column represents may change at any time. Thus traditional spreadsheet reference methods do not apply. In our interface, references to a cell are translated into the tuple

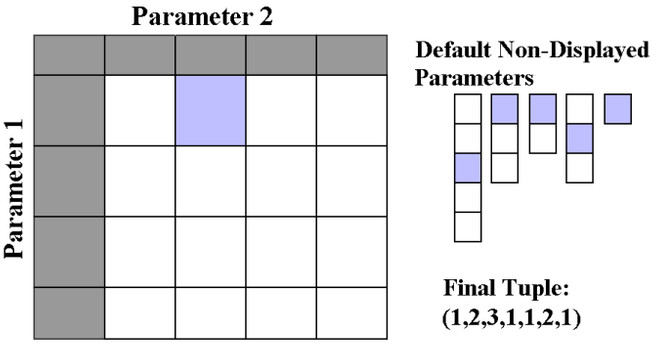


Fig. 8. An example of referencing a cell. A reference to a cell, in this case cell `(1,2)`, is translated into a tuple representing the positions of the cell's parameter values in their respective parameter lists. The translated reference is `(1,2,3,1,1,2,1)`.

identifying that cell in the visualization parameter space. If the second cell in first row is selected, the tuple would reference the second parameter in the list of displayed column parameters, the first parameter in the list of row parameters, and references to the default non-displayed parameters (see Figure 8). A references to a row or column is translated into a parameter reference in a similar manner. When the parameters that are referenced by a tuple are changed, the corresponding result is updated as well. By way of example, consider a data set representing several variables from a turbulent jet simulation (see [20] for more information). In the simulation, two different sums of the variables should represent the same value. Using a user defined function *sumData* to create a new data set by summing the values over the entire volume of its arguments, these two sums can be compared visually using the following script:

```
addParameter(
  "Data Set",
  sumData([column( 0 ), column( 1 )])
render( cell( 1, 3 ) )
addParameter(
  "Data Set",
  loadData( "jet_a3a4a5a6" ) )
render( cell( 1, 4 ) )
```

Figure 9 displays the results of executing this script. These scripts are a versatile way to drive the visualization process.

VI. Encapsulating and Sharing the Visualization Process

The spreadsheet eases collaboration by allowing the exchange of more information than a set of images. With only a set of images, a collaborator has no sense of their order or what parameter values were used to generate them. Expressed as a spreadsheet, the entire visualization process can be communicated to other users. First, the results of the visualization are clearly presented by the spreadsheet. Second, as discussed previously, parameters used for each cell are easily identified. Finally, the history of the process can be viewed, stored, and shared among others.

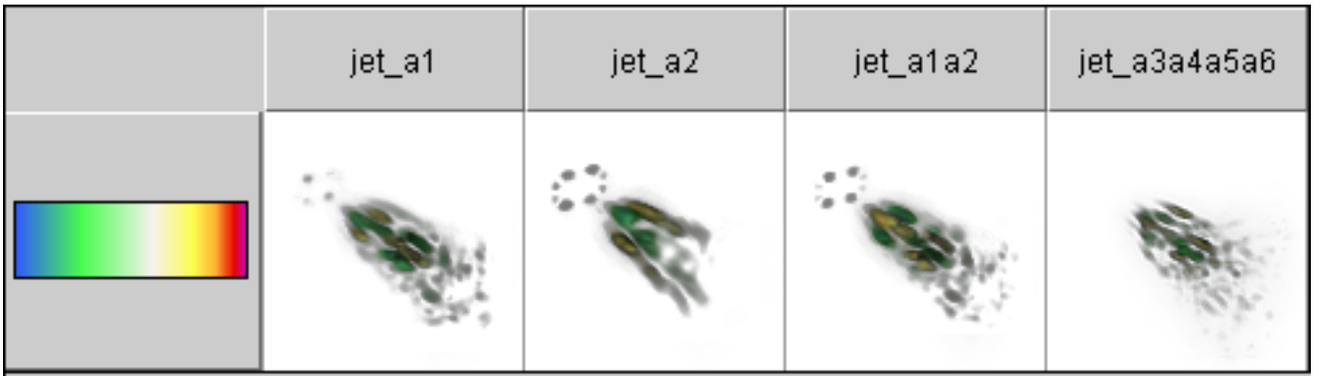


Fig. 9. Another spreadsheet examining multiple data sets. The data represent distinct variables in a multi-variate turbulent jet simulation. The entire simulation has 9 variables. Two of the variables are displayed in the first two columns. The third column is the sum of the first two variables over the entire volume. The fourth column is the sum of four other non-displayed variables. Both sums are supposed to represent the total flow through the jet.

A. History Display

The history of the visualization process can be used to gain insight to where a user has been and where they can further explore. The interface displays the current state of the visualization via its projection of the parameter space. As an option, the user can color the borders of the cell according to the time when the results was generated. Colored borders present the history information at-a-glance: "hotter" borders represent more recently modified cells than "cooler" borders. For a more extensive inspection of the history, the history animation can be viewed.

The history animation technique uses some ideas from Igarashi et al's animation of the relationship between spreadsheet formulas [21]. In their work, they use arrows which fade into view, move from the source of formula data to the cells which use that data, and then fade out. For our history animation, we start from an empty spreadsheet and "fade-in" subsequent changes until the entire state is represented. For transitions between stacks, the old stack fades out as the new stack is displayed. In this manner, the entire history of the visualization process is communicated.

The history animation is controlled through two means. First, during the animation, the user can change the speed of the animation to "fast-forward" through the progress. This fast-forwarding is useful to gloss over portions of the exploration that are irrelevant to a user's presentation. Second, cells can be marked as "important" before the animation begins; only these cells will be presented during the history animation. This capability is especially useful in the context of collaboration. An animation highlighting the salient features of the visualization is more informative than one that displays test images and final results with equal priority.

B. On-Line Collaboration

The spreadsheet can also be used in shared collaboration environments. In this case, the spreadsheet acts as a window into a shared visualization in progress. In one scenario, users work individually, synchronizing parameters and results as desired. In another, changes in the state of the exploration can be communicated concurrently to all users. Both situations can be useful: the former when users are looking for different results and the latter when an expert is driving the exploration.

C. Off-Line Collaboration

To communicate the results of a visualization exploration session, the session must be stored off-line in some manner. This off-line storage format should record everything to recreate the session: the type of visualization performed, the sequence of parameters explored, and the corresponding results. To be effective, the format should not only be understandable by the spreadsheet itself but translatable into formats amenable to data-mining, presentation, and analysis. For these reasons, we have developed an XML-based off-line storage format.

XML [22] is a text-based language for generating descriptions of documents. It is a meta-language used to create languages for documents. There are a wealth of proposed XML technologies to assist in the translation of XML into other formats such as HTML [23] and query its contents [24]. It is thus possible to load results from a spreadsheet visualization session into another visualization system if a translator becomes available.

The off-line representation can be partitioned into two sections: one section representing all the parameters and results explored and another section for representing the visualization process itself. All explored parameters and results—even those later removed from the visualization—are stored in order to recreate the entire session. The results reference the parameters that generated the result and the corresponding visualization (be it an image, embedded VRML model, or other data). The process information is akin to a transaction log. Each element in the list of states represent what changed from the last state to the current state. These "deltas" are built from a set of atomic state change operations. Example atomic operations include adding a parameter value, changing a parameter, removing a parameter, creating a visualization,

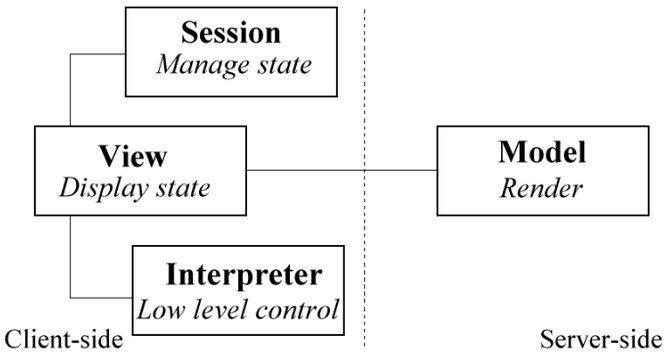


Fig. 10. The visualization spreadsheet framework.

and navigating through the visualization space. These operations reference the parameter or value relevant to the change. As some interactions with the spreadsheet can perform several changes in a single action—for example, applying a script in the the interpreter—the atomic operations can be aggregated when needed.

The two sections of the storage file can be used separately if needed. For example, the deltas used in representing state may not be applicable for storing state in an image graph and may be ignored. Instead, the parameters and results could be used to generate an image graph representation.

Another form of off-line collaboration is the use of spreadsheet templates. Templates are interpreter scripts generated by experts that perform automated manipulation and analysis of the visualization data. For example, a template could generate optimal color and opacity maps after analyzing the input data sets, and display the results in the spreadsheet. Templates can be distributed with data sets to perform initialization or other functions to assist users to understand the data.

VII. System Architecture

We have developed an object-oriented framework which implements the spreadsheet features described in this paper. It consists of four main components: a view object which handles user interaction and displays the spreadsheet, a session object which records changes in the spreadsheet state, a model object which renders the visualization and applies operators, and an interpreter object which executes scripts to manipulate the spreadsheet's state. Figure 10 illustrates the system.

The primary testing platform for the spreadsheet uses direct volume visualization. The design has also been tested in a few other domains. Our current implementation is in Java, using JPython [25], a native Java implementation of the Python language [26], as its interpreter engine.

A. View Object

The view object displays the spreadsheet and handles user interaction. To facilitate its use of distributed environments, the view is local to each client. This prevents local information from being needlessly communicated. The view is a general object. It can be reused in different visualization applications so long as renderers and editors for the parameters and results displays are provided. The view uses the session object to determine the current state of the visualization and the model object to determine what operations it can apply and to generate visualization results.

B. Session Object

The session object manages the state of the exploratory process. It communicates this state to the view object so it can be properly displayed. The state the session object stores is also used to display history information through the color borders and animation discussed previously. It is also the session object's responsibility to store to and load from off-line formats.

As the session object captures all state information for the spreadsheet, it can be used by outside applications to query and modify the progress of the visualization process. This means that the spreadsheet can be used as a history mechanism for another application. This type of "indirect" usage is very powerful: not only can the state of this outside application be shared in a manner previously unavailable, the spreadsheet can be used to apply operators or user defined functions to communicate results back to the original application. This coupling allow users to benefit from the spreadsheet interface while using familiar and pre-existing tools.

Like the view object, the state object is local to the client. If the spreadsheet was later embedded in a web-aware applet, it would be inefficient for it to communicate its state across the network.

C. Model Object

The model object implements the visualization server component of the system. It represents the underlying conceptual model and visualization space. Connected view objects request a result by passing the model a set of parameter values. These results are then stored by the local session object for efficiency. The model could potentially cache previously generated results by their parameter values so requests by different clients for the same result return immediately. How the model actually generates visualization results is a pluggable component. For the volume visualization spreadsheet, the default serial ray casting renderer could potentially be replaced by a renderer that used a cluster of machines (one for each cell) or a module that harnesses the real-time capabilities of specialized hardware such as the VolumePro [27].

The model object also maintains a list of available parameter and value operators. When new operations are added, only the central render server must be updated. At the beginning of a visualization session, clients can request the list of available operations and download the code for them locally, thus saving network communication when the operators are actually applied.

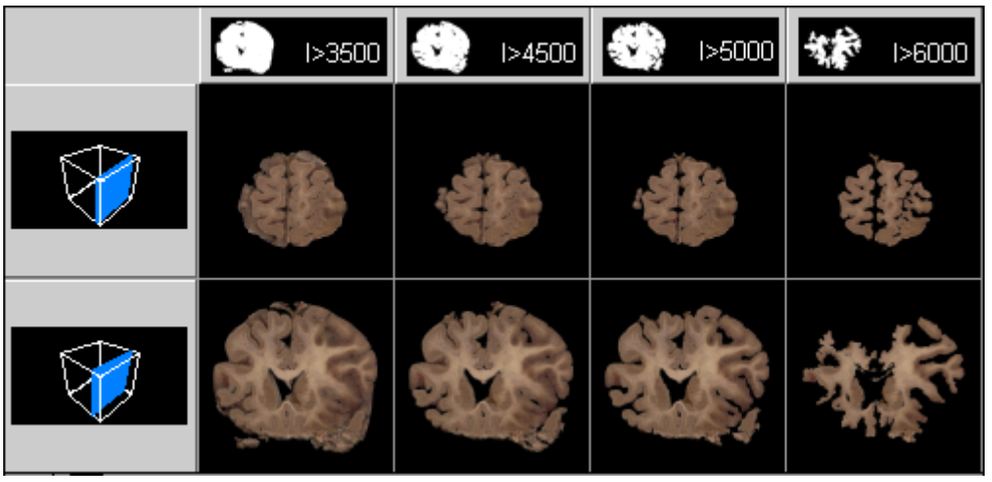


Fig. 11. A spreadsheet analyzing the effect of different parameters on a 3D segmentation pipeline of a human brain. In this example, the effect of color thresholding (the column parameter) is examined across two different slices of the brain (the row parameter).

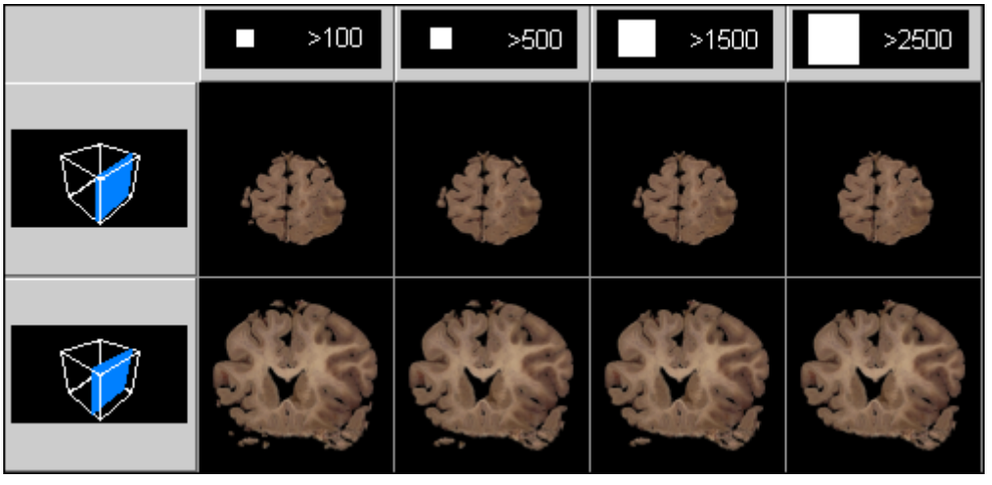


Fig. 12. Another spreadsheet examining the segmentation pipeline. This time, different region size thresholds are displayed along the columns. Note how the effects of the region growing threshold parameter are less pronounced than those from the previous color thresholding stage.

D. Interpreter Object

The interpreter is also locally stored by each client. It implements all the scripting functions described previously. The interpreter allows the user to manipulate the spreadsheet's state in a programmatic way for tasks that would be difficult or awkward using the UI. Generating the view positions from the script in Section V-C would be difficult using the interface but easy with the script. Using the interpreter, users can construct programs to assist them in their exploration.

VIII. Further Examples

We now illustrate the versatility of the spreadsheet-like interface through two additional examples. In the first, the spreadsheet was used to investigate the effect of changing parameters in a 3D segmentation pipeline. The segmentation is applied to a volume image of a frozen human brain. There are six steps in the process, which are controlled by 9 parameters (see [28] for a more complete description). The brain was sliced and photographed. Due to voids in the brain, the images contain information from other slices as well. The purpose of the segmentation is to remove extraneous information not belonging to each slice. The spreadsheet can assist in comparing the effects of the segmentation process on several slices simultaneously.

The first step in the segmentation process applies color thresholding on the image. A particular color component was chosen as it captures the browns in the brain data very well. By changing the threshold value, the effects on the segmentation can be seen (Figure 11). A later step tries to "grow areas" by checking the size of similar regions against another threshold. A comparison of settings of this parameter can be seen in Figure 12. It became clear through this analysis that the initial color thresholding has a greater effect on the segmentation than the later stage and thus should be considered with more care during the segmenta-

tion process. The effects of changing both parameters at the same time was determined by translating the spreadsheet window in visualization space. Using a script to cycle through the color thresholds in Figure 11 (a non-displayed parameter in Figure 12), differences in the images became readily apparent to the eye. Finally, an image difference operator defined on the cells was used to get a quantitative measure of the changes caused by the parameter settings.

One difficulty with the segmentation experiment was the limitation that only two types of parameters can be displayed and edited by the spreadsheet at a time. Creating a result with a specific set of parameters takes significant manipulation. One method of overcoming this shortcoming is to couple the spreadsheet with another application. In this situation, the spreadsheet becomes a type of interactive history mechanism, recording the process of the exploration while the user manipulates the main program. For example, the spreadsheet can communicate with an interactive volume renderer. As the user changes the parameter settings in the volume renderer, they are communicated to the spreadsheet which displays the results. Care must be taken with parameters which vary often and continuously (such as view position in this example). For such parameters, the spreadsheet could be updated at regular intervals or when the user pauses for some time. The default parameters of the spreadsheet always correspond to the currently used parameters in the controlling application. Displayed row and column parameters can be arbitrary—two conventions are to have them be either the most recently modified parameters or the most often modified parameters.

The spreadsheet can remain interactive during this "indirect" modification. The user can switch between exploring with the controlling application or with the spreadsheet. Consider the visualization of a turbulent jet simulation depicted in Figure 13. In this data set, the features of interest are the negative and positive vorticities in the jet. After manipulating the data, the user generated two visualizations which expose the two types of vorticities. Then, by applying a union operator upon the opacity maps of these results, a composite image showing both types of vorticities is generated and displayed in the original program. Exploration can continue from here, using the spreadsheet for more structured control as the session continues.

IX. Conclusions

By visually organizing the data exploration process while providing tools to build upon and share this process our spreadsheet-like interface makes visualization more efficient and effective. The space of visualization parameters is made clear by the spreadsheet's structure and iconic display. The dependence of a result on its parameters becomes transparently available. Previous results can be extended by operators to further discovery. The interpreter can be used to construct complex visualizations in a programmatic manner. Finally, the interface makes the history of the process available to both the user and their collaborators. Combined, these capabilities utilize the inherent iterative nature of the visualization process to a user's advantage.

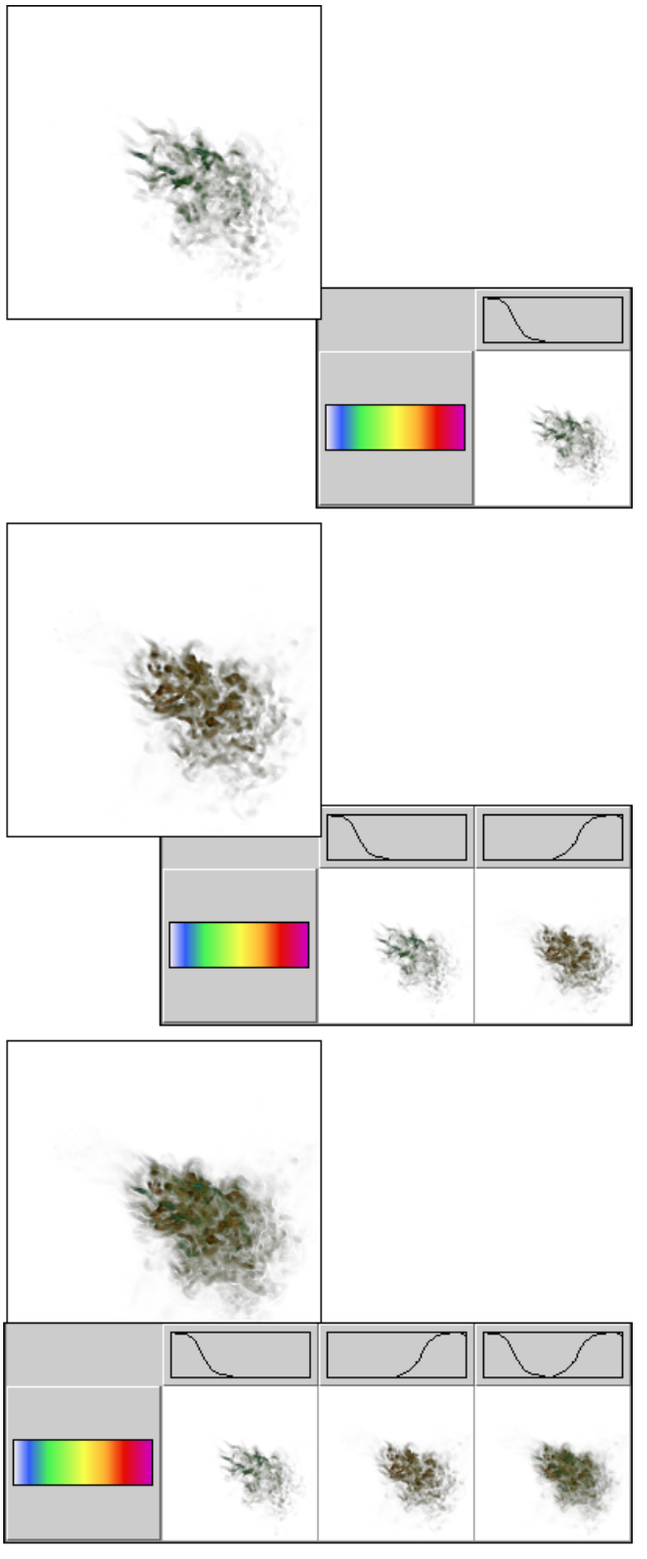


Fig. 13. The spreadsheet can be coupled with an auxiliary program that can control the visualization. As the exploration progresses, the main application updates the spreadsheet with its current state. This interactive history mechanism can also be used to communicate the other way. Here the user applied a parameter operator to the opacity maps (columns) to highlight both the negative and positive vorticities in a turbulent jet data set.

This work is powerful because it is general: it can be applied to a wide domain of visualization problems. The parameter space abstraction can be used in many visualization tasks. The stored representation can be shared and extended in several different ways. These ideas will assist users of visualization in all disciplines to explore, communicate, and understand their results.

A. Future Work

There exist several directions for future research in both the spreadsheet-like interface and visual representations of data exploration in general. Graphics researchers are already familiar with the difficulties involved in navigating a 3D environment with a 2D interface. The spreadsheet complicates matters as it represents a multidimensional space. Our current interface possesses means for setting the row and column parameters. It does not have any method for locating a previously generated image. Consequently, it would also be beneficial to display navigational landmarks that help a user locate themselves in visualization space. It may also be possible to perform queries upon the representation to find parameters or results of interest; the types of queries described by Henze [29] could be used to navigate and explore the visualization space.

There is significant potential research in modeling the visualization process. First, it may be possible to create a general representation of this process. Such a model would have to be able to determine the core features of the visualization parameter space and how the user can interact/explore this space. The representation of the "path" through visualization space can be enhanced as well by moving from a linear trace to something that captures the branching nature of the exploration (as in [30]). Given such a formalism, an XML off-line format similar to the one here could be created to be shared and built-upon by collaborators using a variety of tools. This framework can also be enhanced by including extensible meta-data to annotate the visualization. User studies comparing the developed interfaces to current interfaces would also assist in addressing current interface weaknesses. A better understanding of the visualization process can only help systems designers create interfaces more attuned to their users needs.

Acknowledgments

This work was supported by NASA Ames Research Center through an NRA award under contract NAG2-1216, the National Science Foundation under contracts 9983641 (CAREER Awards) and ACI 9982251 (LSSDSV program), and Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreements B347878 and B503159. Thanks go to Ayodeji Demuren, Lawrence Berkeley National Laboratory, Arthur Olson, Philip Smith, Arthur Toga, Robert Wilson, and the Visible Human Project for the data sets. Michael Gertz at UC Davis assisted in the development of the XML model. The authors thank the IEEE Visualization 2000 reviewers and members of the UC Davis Visualization and Graphics Group for their input and assistance.

References

[1] Ed H. Chi, John Riedl, Phillip Barry, and Joseph Konstan, "Principles for information visualization spreadsheets," *IEEE Computer Graphics & Applications*, vol. 18, no. 4, pp. 30–38, July - August 1998.

[2] Sherry Yang, Margaret M. Burnett, Elyon DeKoven, and Moshé Zloff, "Representation design benchmarks: A design-time aid for VPL naviable static representation," *Journal of Visual Languages and Computing*, vol. 8, no. 5/6, pp. 563–599, October - December 1997.

[3] Craig Upson, Thomas A. Faulhaber, Jr., David Kamins, David Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam, "The Application Visualization System: a computational environment for scientific visualization," *IEEE Computer Graphics and Applications*, vol. 9, no. 4, pp. 30–42, July 1989.

[4] Mark Young, Danielle Argiro, and Steven Kubica, "Cantata: Visual programming environment for the Khoros system," *Computer Graphics*, vol. 29, no. 2, pp. 22–24, May 1995.

[5] Greg Abram and Lloyd A. Treinish, "An extended data-flow architecture for data analysis and visualization," *Computer Graphics*, vol. 29, no. 2, pp. 17–21, May 1995.

[6] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber, "Design galleries: A general approach to setting parameters for computer graphics and animation," in *Proceedings of SIGGRAPH97*. ACM SIGGRAPH, Aug. 1997, pp. 389–400.

[7] Kwan-Liu Ma, "Image graphs - a novel approach to visual data exploration," *IEEE Visualization '99*, pp. 81–88, October 1999.

[8] Ivan Herman, Guy Melançon, and M. Scott Marshall, "Graph visualization and navigation in information visualization: A survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 1, pp. 24–43, January-March 2000.

[9] Margaret M. Burnett, "Visual programming," in *Encyclopedia of Electrical and Electronics Engineering*, John G. Webster, Ed. John Wiley and Sons Inc., New York, 1999.

[10] Margaret Burnett and Allen Amber, "Interactive visual data abstraction in a declarative visual programming language," *Journal of Visual Languages and Computing*, vol. 5, no. 1, pp. 29–60, March 1994.

[11] Paul Carlson, Margaret Burnett, and Jonathan Cadiz, "A seamless integration of algorithm animation into a declarative visual programming language," in *Proceedings Advanced Visual Interfaces (AVI'96)*, May 1996.

[12] Margaret Burnett, Andrei Sheretov, and Gregg Rothermel, "Scaling up a "what you see is what you test" methodology to spreadsheet grids," in *Proceedings of IEEE Symposium on Visual Languages 1999*. IEEE, Sept. 1999.

[13] Marc Levoy, "Spreadsheets for images," *Proceedings of SIGGRAPH 94*, pp. 139–146, July 1994.

[14] A. F. Hasler, K. Palaniappan, and M. Manyin, "A high performance interactive image spreadsheet (IISS)," *Computers in Physics*, vol. 8, pp. 325–342, May - June 1994.

[15] K. Palaniappan, A. F. Hasler, J. Fraser, and M. Manyin, "Network-based visualization using the distributed image spreadsheet (DISS)," in *Seventeenth Int. Conf. on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography and Hydrology*, Albuquerque, NM,, Jan. 14-19, 2001, American Meteorological Society.

[16] T.J. Jankun-Kelly and Kwan-Liu Ma, "A spreadsheet interface for visualization exploration," in *Proceedings of IEEE Visualization 2000*, Salt Lake City, Oct. 2000, IEEE.

[17] Robert Spence, *Information Visualization*, p. 92, ACM Press, 2001.

[18] Greg Schussman, Kwan-Liu Ma, Davis Schissel, and Todd Evans, "Visualizing DIII-D Tokamak magnetic field lines," in *Proceedings of IEEE Visualization 2000*, Salt Lake City, Oct. 2000, IEEE.

[19] Patrick J. Moran and Chris Henze, "Large field visualization with demand-driven calculation," in *Proceedings of IEEE Visualization 1999*, David Ebert, Markus Gross, and Bernd Hamann, Eds., New York, Oct. 1999, pp. 27–34, ACM Press.

[20] Robert V. Wilson and Ayodeji O. Demuren, "On the origin of streamwise vorticity in complex turbulent jets," in *Proceedings of ASME Fluids Engineering Division Summer Meeting (FEDSM98)*. ASME, 1998.

[21] Takeo Igarashi, Jock D. Mackinlay, Bay-Wei Chang, and Polle T. Zellweger, "Fluid visualization of spreadsheet structures," in *Proceedings of IEEE Symposium on Visual Languages 1998*. IEEE, Sept. 1998.

[22] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0," Tech. Rep., World Wide Web Consortium, 2000, http://www.w3.org/TR/2000/REC-xml-20001006.

[23] World Wide Web Consortium, "Extensible Stylesheet Language (XSL)," Tech. Rep., World Wide Web Consortium, 2000, http://www.w3.org/TR/xsl/ (Work in Progress).

[24] World Wide Web Consortium, "XQuery: A Query Language for XML," Tech. Rep., World Wide Web Consortium, 2001, http://www.w3.org/TR/xquery/ (Work in Progress).

[25] Jim Hugunin, "Python and Java: The best of both worlds," in *Proceedings of the 6th International Python Conference*. CNRI, 1997, http://www.python.org/workshops/1997-10/-proceedings/hugunin.html.

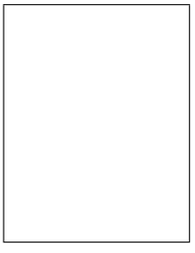
[26] Guido van Rossum, *Python Language Reference Manual*, July 1999, http://www.python.org/doc/ref/ref.html.

[27] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler, "The VolumePro real-time ray-casting system," in *Proceedings of SIGGRAPH99*, Alyn Rockwood, Ed., N.Y., Aug. 8–13 1999, ACM SIGGRAPH, pp. 251–260, ACM Press.

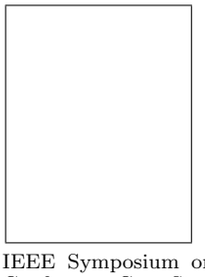
[28] Ikuko Takanashi, Eric Lum, Joerg Meyer, Kwan-Liu Ma, Bernd Hamann, and Art Olson, "Segmentation and volume rendering of human brain cryosections," Submitted to IEEE Visualization 2001 Case Studies.

[29] Chris Henze, "Feature detection in linked derived spaces," in *Proceedings of the IEEE Visualization 1998*, New York, Oct. 18–23 1998, pp. 87–94, ACM Press.

[30] John Peter Lee and George G. Grinstein, "An architecture for retaining and analyzing visual explorations of databases," in *Proceedings of Visualization 1995*. 1995, pp. 101–108, IEEE.



T.J. Jankun-Kelly is a PhD Candidate and graduate student researcher in computer science at the University of California, Davis. His interests include scientific visualization, information visualization, computer graphics, and theory. He received his BS in 1997 from Harvey Mudd College and MS in 1999 from the University of California, Davis. He expects his PhD in late 2001. Contact Jankun-Kelly by e-mail at tjk@acm.org.



Kwan-Liu Ma is an associate professor of computer science at the University of California, Davis, where he teaches and conducts research in the areas of computer graphics and scientific visualization. His career research goal is to improve the overall experience and performance of data visualization through more effective user interface designs, interaction techniques, and high-performance computing. Ma received his PhD from the University of Utah in 1993. He served as co-chair for the 1997 IEEE Symposium on Parallel Rendering, the IEEE Visualization Conference Case Studies in 1998 and 1999, and the first NSF/DOE Workshop on Large Data Visualization. Contact Ma by e-mail at ma@cs.ucdavis.edu.