Title
A Spreadsheet Interface for Visualization Exploration

Authors
Jankun-Kelly, T. J.
Ma, Kwan-Liu

Publication Date
2000

Peer reviewed

As printed in IEEE Visualization 2000

A Spreadsheet Interface for Visualization Exploration

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

Abstract

As the size and complexity of data sets continues to increase, the development of user interfaces and interaction techniques that expedite the process of exploring that data must receive new attention. Regardless of the speed of rendering, it is important to coherently organize the visual process of exploration: this information both grants insights about the data to a user and can be used by collaborators to understand the results. To fulfill these needs, we present a spreadsheet-like interface to data exploration. The interface displays a 2-dimensional window into visualization parameter space which users manipulate as they search for desired results. Through tabular organization and a clear correspondence between parameters and results, the interface eases the discovery, comparison and analysis of the underlying data. Users can utilize operators and the integrated interpreter to further explore and automate the visualization process; using a method introduced in this paper, these operations can be applied to "cells" in different "stacks" of the interface. Via illustrations using a variety of data sets, we demonstrate the efficacy of this novel interface.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces—Graphical user interfaces (GUI); H.5.3 [Information Interfaces and Presentation]: Group and organization Interfaces—Collaborative computing; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques

Keywords: spreadsheets, user interfaces, knowledge representation, scientific visualization, visualization systems, volume rendering

1 Introduction

Without effective interfaces to represent the process and results of visualization, a user cannot properly utilize the underlying techniques to extract information from their data. While research in visualization has been driven by the need to view more realistic and informative visualizations of very large data sets in a reasonable amount of time, few efforts have been devoted to storing and presenting the data exploration process itself. This information can be shared and reused, enabling users to build upon past work without repeating themselves. Toward this end, this paper describes a spreadsheet-like interface for data exploration and demonstrates its functionality through a set of examples.

During the data exploration process, a user attempts to discover a set of parameter values to create desired visualizations; it is an interface's task to make the search and manipulation of these values as transparent as possible. Our spreadsheet-like interface provides the following capabilities to assist in this task:

- A tabular structure that encapsulates the visualization.
- Operators upon parameters and results to analyze and manipulate the visualization.
- An interpreter that can control the visualization process at a lower level.

These features help the user in their work, and make it easier to share information with collaborators. The table provides more context than manipulation of parameters or results alone. Operators efficiently create new results from previous values. Finally, the interpreter allows experts to perform complex operations upon the data that supersede the facilities provided by the UI.

Numeric spreadsheets have been used for some time; the interface has also been applied to other domains. The visual language community has extensively studied spreadsheets and their applications; for example, the Forms/3 system [2] has been used to discuss spreadsheet animation, dynamically expanding grids, and psychological factors in designing user interfaces [4, 3, 18]. In graphics, the SI system [11] wraps a spreadsheet around a general image processing kernel; users manipulate scripts to generate cell values. Hasler, et al. [7] also describe a spreadsheet system for image manipulation in the context of satellite data analysis. Unlike our spreadsheet, both of these systems focus upon data display and manipulation rather than exploration. Chi, et al. [5] describe a set of principles for visualization spreadsheets used by their SIV system. While we adopt some of their principles, our system is designed specifically to ease the visualization exploration process. Our spreadsheet displays a movable window into visualization space while previous work collapses the entire space into 2D. This windowing of visualization space allows quick data exploration. The spreadsheet can also be used as a representation of the visualization as a whole; where conventional spreadsheet are driven by direct user interaction, our spreadsheet can react to changes in the visualization itself.

2 Data Exploration Techniques

Visualization provides insight, however the process of visualization—that of data exploration—can be enlightening as well. Without a means to document the effort spent generating visualizations, these efforts are lost once the process is complete. In cases where generating the visualization is computationally expensive or the correspondence between parameter values and their results are non-intuitive, capturing the exploration process

*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

is especially important. Thus, user interfaces to data exploration should not only create visualizations, but store the history of the exploration in a manner that allows exchange and manipulation. We propose that such an interface should meet the following criteria:

- Allow the user to set and manipulate parameter values to generate visualizations.
- Display the relationship between and context of different visualizations.
- Provide a set of parameter and value operators to extract information and generate new visualizations.
- Encapsulate the history of the exploration process for collaboration.

This paper demonstrates that our interface satisfies all of these criteria.

In traditional interfaces, a user iteratively changes parameter values in order to search for the desired result. This trial and error process is inefficient and does not provide context that directs a user toward their goal. Automatic systems that generate parameter values, such as Kindlmann and Durkin's transfer function generation technique for volume visualization [10], can help this process, but their result is lost once a user modifies the parameter again. Once an acceptable visualization 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 interfaces do not allow sophisticated control of the exploration.

Several commercial systems use a data-flow interface for visualization [15, 19, 1]. These systems provide a visual programming environment that allows a user to generate a directed graph representing the flow of data through the system; the network can be shared, allowing for more powerful collaboration than simple interfaces. This interface is more suited to generating visualization results than recording the data exploration process: the history of the network is not recorded and there are no operations to manipulate the network to create new networks.

The Design Galleries system [13] 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 provides 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 [12] was designed with visualization data exploration in mind, and thus satisfies the criteria we presented. An image graph is a dynamic 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 with collaborators, thus providing a history of the final result with the result itself. There are some weaknesses with the image graph approach. As exploration progresses, the graph can become too large to display effectively. In addition, applying operations to several different related paths of images is difficult; this is especially true for graphs displaying different data sets—there are no paths between them in the graph. These concerns are addressed by our spreadsheet-like representation.

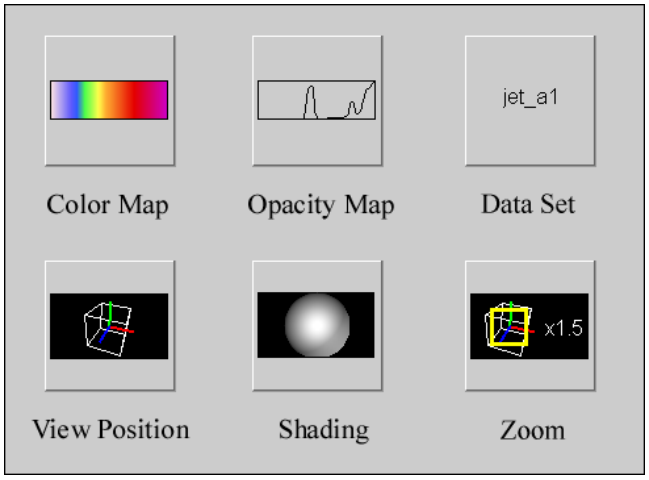

Figure 1: Visual representation of some parameters displayed by the spreadsheet. As a user edits the underlying parameter, the icon of the parameter is updated.

3 Spreadsheet-based Exploration

Like the Design Galleries and image graph systems, we consider visualization exploration a process of examining a multi-variable space of parameter values. Each n-tuple in the space represents a combination of parameters that produce a visualization. For the purposes of this discussion, we will describe a spreadsheet interface specialized for volume rendering, though the ideas can be applied in other domains. The points in the visualization space are the volume rendered images specified by the combined parameter values. The parameters our system currently utilizes are data sets, color maps, opacity maps, view position, shading coefficients, sample size, and image size/zoom; when selected as a row or column parameter, they are displayed as in Figure 1.

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 more complicated to display. Our spreadsheet is a movable, scalable window into space; by manipulating the visualization parameters, the user changes the position and size of this window. The spreadsheet displays a planar projection spanned by two axes of the visualization space; for example, the rows could represent opacity maps while the columns could display color maps as in Figure 2. For the non-displayed parameters, the user selects a set of values to be used as defaults. A cell in the spreadsheet is identified by combining these defaults with the parameter values corresponding to the cell's row and column indices. The spreadsheet window is translated by changing one of the default values for a non-displayed parameter; the cells are automatically updated. For example, Figure 2 illustrates what occurs when the view position, a non-displayed parameter, is changed—the spreadsheet's position in visualization space changes as well. The spreadsheet window can also be rotated, as Figure 3 demonstrates. Rotation occurs when two new parameter values are chosen for display after the user selects new default values for the previously displayed parameters; the window is rotated about the cell with the selected default values. In the figure, default values for the color and opacity map parameters were chosen by selecting the image combining the parameters. View position and zoom factor were then selected as the new displayed parameters; the previously selected cell appears in the updated spreadsheet as well. Thus, the visualization process becomes a process of maneuvering the spreadsheet window through

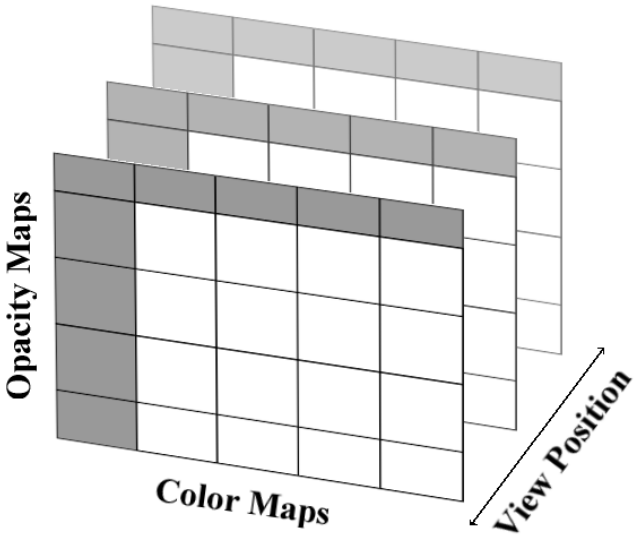

Figure 2: Our spreadsheet is a view of two dimensions of a visualization space; in this example, opacity maps are displayed along rows and color maps along columns. A particular cell is rendered by combining the non-displayed parameters' default values with the parameter values corresponding to the row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

the parameter space.

Previous visualization spreadsheets collapse visualization space into 2D without constraining what values were 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.

3.1 Static Capabilities

After starting the spreadsheet, a user selects the initial data set to visualize; default values are used for the other parameter values. Using the spreadsheet, the user can select which parameters to display along the rows and columns, and add, remove and position column and row values as desired. Only requested cells are rendered, avoiding the overhead of rendering the entire table. 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 identifies which parameters correspond to an image by rendering the row and column labels as in Figure 1.

Figure 4 demonstrates a spreadsheet-driven visualization; in this case, the user wished to display separate skin and bone surfaces for a foot medical data set. 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

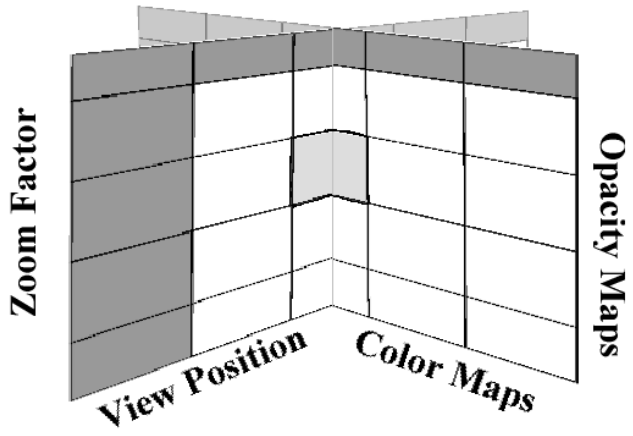

Figure 3: The spreadsheet window can also be rotated in visualization space. For example, starting from a sheet displaying color and opacity maps, the user first selects an image with the desired properties; these two parameters will become the new default values. By then selecting two new parameters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

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 spreadsheet 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; for example, Figure 5 displays a sequence of data sets representing time steps of a turbulent flow 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, for example. In Figure 6, two variables from a multi-variate turbulent jet simulation are displayed along with their sum in the first three columns; the fourth column displays the sum of four other flow variables. The user can visually compare the two sums, which represent different representations of the total flow (see [17] for more information). Again, the user can further compare the values by modifying the rendering parameters. The tabular structure also suggests natural parallelism when applying the operations from the next section; if a new column is generated by an operation, each of the new cell values could be distributed to separate processors to be rendered.

3.2 Dynamic Capabilities: Parameter and Value Operations

Most spreadsheets define a set of operations that can be applied to the cells; for example, numerical spreadsheets provide functions for algebraic manipulation and statistical analysis. 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: for example, set operations can be applied to color and opacity maps, view

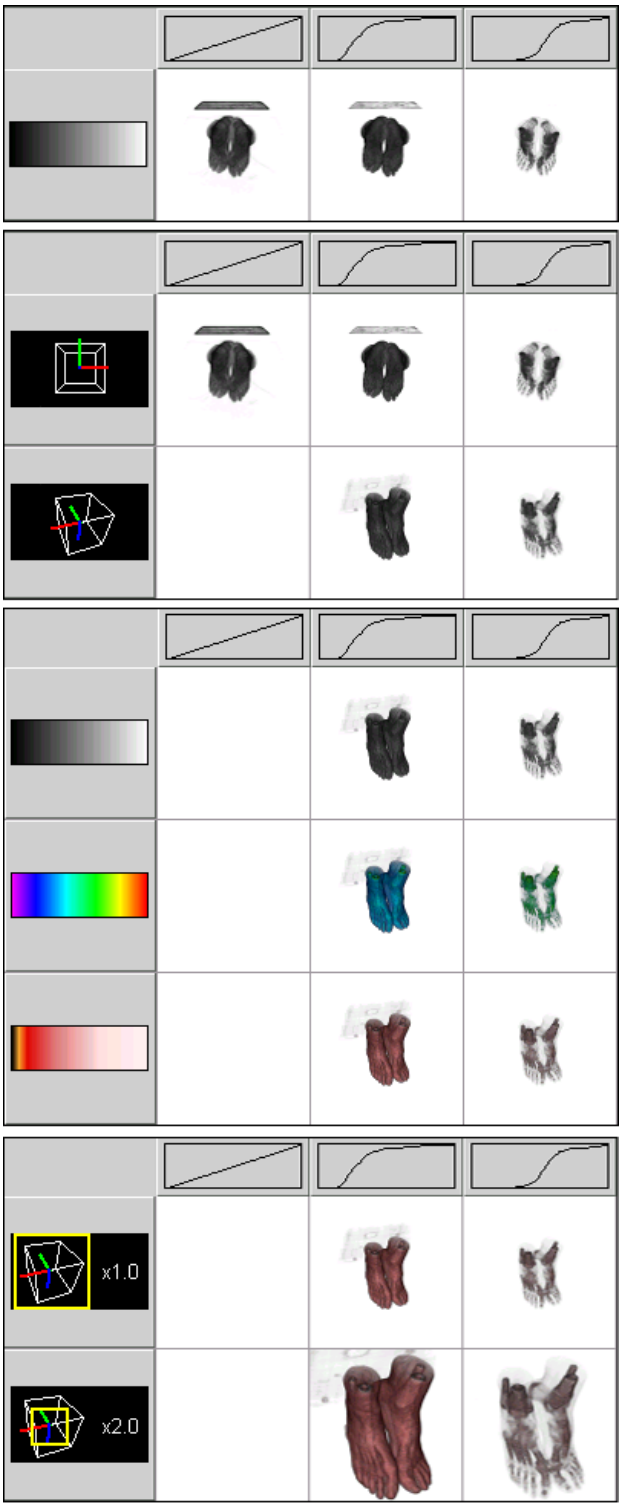

Figure 4: A sequence of spreadsheets displaying the visualization of a foot data set; blank cells represent non-rendered images. The goal was to compare skin and bone surfaces. The user first determined appropriate opacity maps before modifying the view position, color map and zoom factor. The spreadsheet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

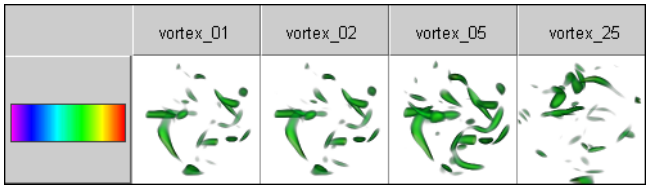

Figure 5: A spreadsheet displaying multiple data sets representing time steps in a vortex simulation. Modifying the displayed color map would change all the data set images at once, a task that would be more difficult in other representations.

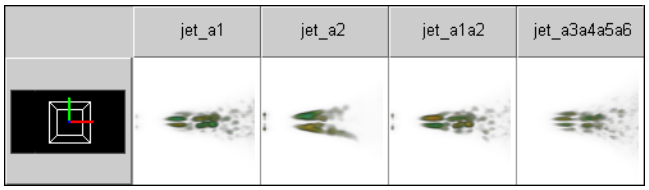

Figure 6: Another spreadsheet examining multiple data sets; the data represent distinct variables in a multi-variate turbulent jet simulation. The entire simulation has 9 variables; the first two columns are two of the variables. The third column is the sum of the first two variables over the entire 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.

positions can be interpolated and histograms can be derived from data sets. To apply a parameter operator, a user first selects a range of column or row values; these will be used as the operator's arguments. The user then selects an operator to apply and, if necessary, customizes its behavior. For example, Figure 7 demonstrates the union operator applied to opacity and color maps to display two separate features (positive and negative vorticity of the flow, respectively) of a turbulent flow simulation data set together; the new maps are automatically added to the table.

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 multidimensional, cells from alternate "stacks" of the spreadsheet can be selected at the same time. For example, 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 (Figure 8), the user could follow these steps:

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

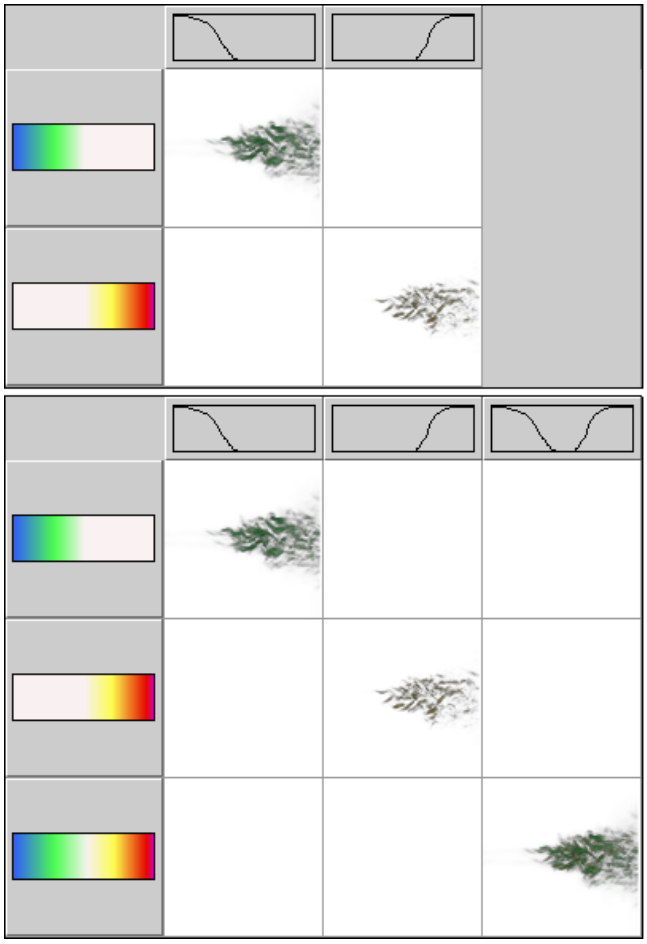

Figure 7: An example of applying parameter operators, in this case the union of color and opacity maps. The two original maps represent negative and positive vorticity, respectively, in a turbulent flow data set; the final image displays them together.

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.

3.3 Animation

Unlike static images, animations better display 3-dimensional features of data over both spatial and temporal domains; thus, it is important for our interface to support animation creation. 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 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 8. 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 resulting animation would illustrate the structure of the flow along a path of continuously varying viewpoints.

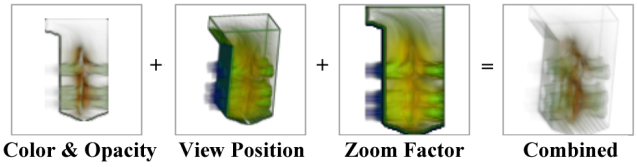

Figure 8: Selected cells for a composition operator; in this case, the user wishes to use the color and opacity maps of the first image, the view position of the second and the zoom factor of the third. The fourth image displays the desired visualization.

Animation is also used for another purpose: illustrating the history of the visualization. This technique uses some ideas from Igarashi et al's animation of the relationship between spreadsheet formulas [9]. 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.

3.4 Scripting

Another dynamic capability of the spreadsheet is the use of an interpreter. The interpreter can access any parameter or cell value or operator and is not limited to the values currently displayed. Non-selected parameter values can be manipulated and new images rendered. 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 programmatically.

Our implementation of scripting differs from common macro languages in numerical spreadsheets or the scripting abilities in the spreadsheets described by Chi, et al. [5]. In these applications, cells are referenced by their row and column values. If a cell's value changes, all formulas which reference that cell's row and column are updated; if other formulas depend on the changed formula, they are updated as well. In our spreadsheet, the values used for rows and columns can change dynamically; the parameter a row or column represents may change at any time. Thus traditional spreadsheet reference methods do not apply. Currently, a reference to a cell location is translated into a tuple representing the positions of that cell's parameter values in the parameter lists. Thus, if the

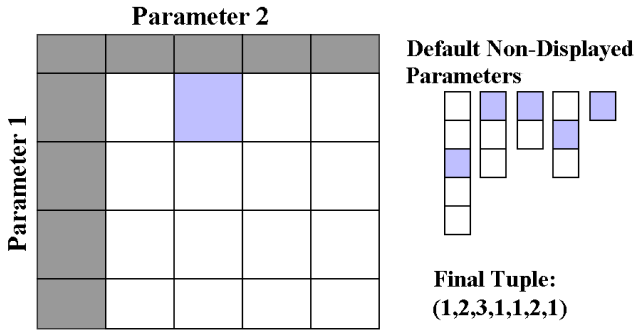

Figure 9: An example of referencing a cell. A reference to a cell, in this case cell `(1,2)`, is translated into a tuple representing the positions of the cell's parameter values in their respective parameter lists. The translated reference is `(1,2,3,1,1,2,1)`.

second cell in the first row is selected, the tuple elements for the parameters currently displayed in the row and column would be one and two, respectively, with appropriate values for the non-displayed parameters (see Figure 9). References to rows or columns are translated to indices into the appropriate parameter lists. For example, the following script performs the same operations used to create Figure 7; if the user changes one of the original color or opacity maps, the derived cell will be automatically updated:

```
addParameter( "Opacity Map",
              union( column( 1 ),
                     column( 2 ) ) )
addParameter( "Color Map",
              union( row( 1 ),
                     row( 2 ) ) )
render( cell( 3, 3 ) )
```

4 Collaboration

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 communicated through coloring the cell borders and animating the spreadsheet. Colored borders present the history information at-a-glance; "warmer" (lighter) borders represent more recently modified cells than "cooler" (darker) borders. The animation technique discussed in Section 3.3 provides a more in-depth look at the history. The information the spreadsheet displays can be used to understand the source of the images, or used to derive new results; without it, the process would have to be repeated for each new user.

Besides their off-line usage, our spreadsheet interface can be used on-line as well. Several active views can access the same visualization in progress. Either the users work individually, synchronizing parameters as desired, or the system maintains a pool of parameter values that the users share—as new parameter values are added, all users would see the result. Both situations could be useful: the former when users are looking for different results and the latter when an expert is driving the exploration.

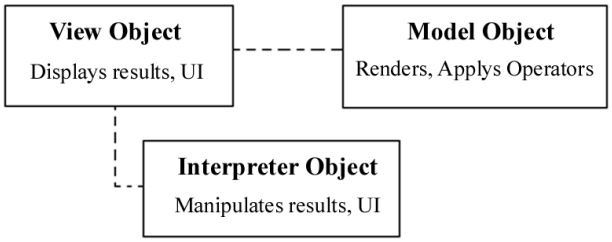

Figure 10: The visualization spreadsheet framework.

Another form of collaboration is the use of spreadsheet templates. Templates are 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 understand the data.

5 System Architecture

We have developed an object-oriented framework which implements the spreadsheet features described in this paper. It consists of three main components: a view object which handles user interaction and displays the spreadsheet, 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 system described in this paper implements a volume visualization spreadsheet using the developed framework; however, the framework itself can be applied to different types of visualization. Our current implementation is in Java, using JPython [8], a native Java implementation of the Python language [16], as its interpreter engine.

5.1 View Object

The view object displays the spreadsheet, handles user interaction, and manages the state for the local spreadsheet object. It is important that the view manages the spreadsheet state; if the view object is later embedded in a web-aware applet, it would be inefficient for it to communicate its state across the network.

5.2 Model Object

The model object implements the visualization server component of the system. For our volume visualization spreadsheet, it implements a ray casting algorithm for the rendering. Connected view objects request an image by passing the model a set of parameter values; the model caches previously generated images by their parameter values so requests by different clients for the same image return immediately. Though our current implementation uses a simple, serial ray caster, the renderer is another pluggable component in the system: it can be easily replaced with an object that farms the rendering out to a cluster of machines, one that adapts the rendering based upon the data set and parameter values or module that harnesses the real-time capabilities of specialized hardware such as the VolumePro [14].

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.

The model is responsible for capturing the visualization as a whole. Thus, changes to the model propagate to all connected views. This is important because the model can potentially be modified by outside applications. For example, the model object can be used by a separate image graph; new parameters added by the image graph would be visible to the spreadsheet. Thus, the spreadsheet provides a representation of the visualization through the model.

5.3 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 3.4 would be difficult using the interface but easy with the script. Using the interpreter, users can construct programs to assist them in their visualizations.

6 Conclusions and Future Work

Our spreadsheet interface assists in the visualization process in two ways. First, the structure of the spreadsheet provides an organized means of exploring the space of visualization parameters. In volume visualization, it is not immediately apparent what rendering parameters correspond to the displayed image; the spreadsheet clearly identifies which parameters belong to which images. By understanding how the result depends on parameter values, a user can quickly navigate the space of parameters towards the desired image. Since generating the images can be time consuming, streamlining the process makes the entire session more efficient. The structure also eases collaboration as it communicates all the steps used to find the desired results; this history can be useful in exploring alternate paths at a later time.

The dynamic capabilities are the second property that assist in visualization. They speed the search for parameter values by allowing users to generate new parameters by combining a range of older values; they provide insight by analyzing the explored results. The structure of the spreadsheet allows operators to be applied to a wider range of values than in other interface designs. Experts can work very efficiently by using the interpreter to manipulate visualizations directly.

This work raises some interesting questions regarding visualization interfaces. Though the interface was designed for exploration purposes, it can be argued that it also provides a useful representation of the visualization as a whole. Regardless of the actual interface used, our spreadsheet can encapsulate the visualization's history. Thus, one can imagine an application where the user directly manipulates the visualization in a separate area; as new parameter values are tried, their results would be automatically added to the spreadsheet. Similarly, instead of editing just the column and row values, the user might also manipulate the objects in the cells; again, new parameters and results would automatically be added to the spreadsheet. Both deserve further study.

6.1 Future Work

There are still some outstanding research areas in our spreadsheet design. 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 provides means for setting the row and column parameters; it does not provide 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. One idea is to use an image graph for navigation purposes, as it can display summary information at a glance; the system would dynamically translate between the two representations. Some preliminary work in this area is positive.

Another open problem for our spreadsheet design is off-line storage. A simple, specialized format that stores which parameters were explored and what images were rendered is sufficient for a simple application. Yet, as central databases of visualization data become more prevalent, the interface between the spreadsheet's data and these repositories becomes important. Meta-data such as the order of the generated visualizations should be stored in some manner as well; different visualization domains have their own meta-data as well. One solution would be an XML [6] specification for storing the spreadsheet data; this would enable the data to be exchanged with different applications. The view could automatically convert its state into XML and communicate it to the model; in this case, the model could be a proxy between the spreadsheet application and a central data repository which understood XML.

In the future, we would like to apply the spreadsheet framework to a variety of visualization domains, such as vector visualization. In addition, we would like to perform a formal user study to measure and compare the effectiveness of current interfaces. These studies could be used to create additional interfaces to address current weaknesses and may lead to the development of a formal specification for interfaces to visualization.

Acknowledgments

This work was supported by the National Science Foundation under contracts 9983641 (CAREER Awards) and ACI 9982251 (LSSDSV program), the Office of Naval Research under contract N00014-97-1-0222, the Army Research Office under contract ARO 36598-MA-RIP, NASA Ames Research Center through an NRA award under contract NAG2-1216, Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159, and the North Atlantic Treaty Organization (NATO) under contract CRG.971628 awarded to the University of California, Davis. We also acknowledge the support of Chevron, General Atomics, and SGI. Special thanks go to Ayodeji Demuren, Philip Smith, Robert Wilson, and the Visible Human Project for the test data sets. Suggestions made by the reviewers and members of UC Davis Visualization and Graphics Group helped improve the final manuscript.

References

[1] Greg Abram and Lloyd A. Treinish. An extended data-flow architecture for data analysis and visualization. *Computer Graphics*, 29(2):17–21, May 1995.

[2] Margaret Burnett and Allen Amber. Interactive visual data abstraction in a declarative visual programming language. *Journal of Visual Languages and Computing*, 5(1):29–60, March 1994.

[3] 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, September 1999.

[4] Paul Carlson, Margaret Burnett, and Jonathan Cadiz. A seamless integration of algorithm animation into a declarative vi-

sual programming language. In *Proceedings Advanced Visual Interfaces (AVI'96)*, May 1996.

[5] Ed H. Chi, John Riedl, Phillip Barry, and Joseph Konstan. Principles for information visualization spreadsheets. *IEEE Computer Graphics & Applications*, 18(4):30–38, July - August 1998.

[6] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. Technical report, February 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

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

[8] 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.

[9] 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, September 1998.

[10] Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *IEEE Symposium on Volume Visualization*, pages 79–86. IEEE, ACM SIGGRAPH, 1998.

[11] Marc Levoy. Spreadsheets for images. *Proceedings of SIGGRAPH 94*, pages 139–146, July 1994.

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

[13] 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*, pages 389–400. ACM SIGGRAPH, August 1997.

[14] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The VolumePro real-time ray-casting system. In Alyn Rockwood, editor, *Proceedings of SIGGRAPH99*, pages 251–260, N.Y., August 8–13 1999. ACM SIGGRAPH, ACM Press.

[15] 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*, 9(4):30–42, July 1989.

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

[17] 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.

[18] 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*, 8(5/6):563–599, October - December 1997.

[19] Mark Young, Danielle Argiro, and Steven Kubica. Cantata: Visual programming environment for the Khoros system. *Computer Graphics*, 29(2):22–24, May 1995.

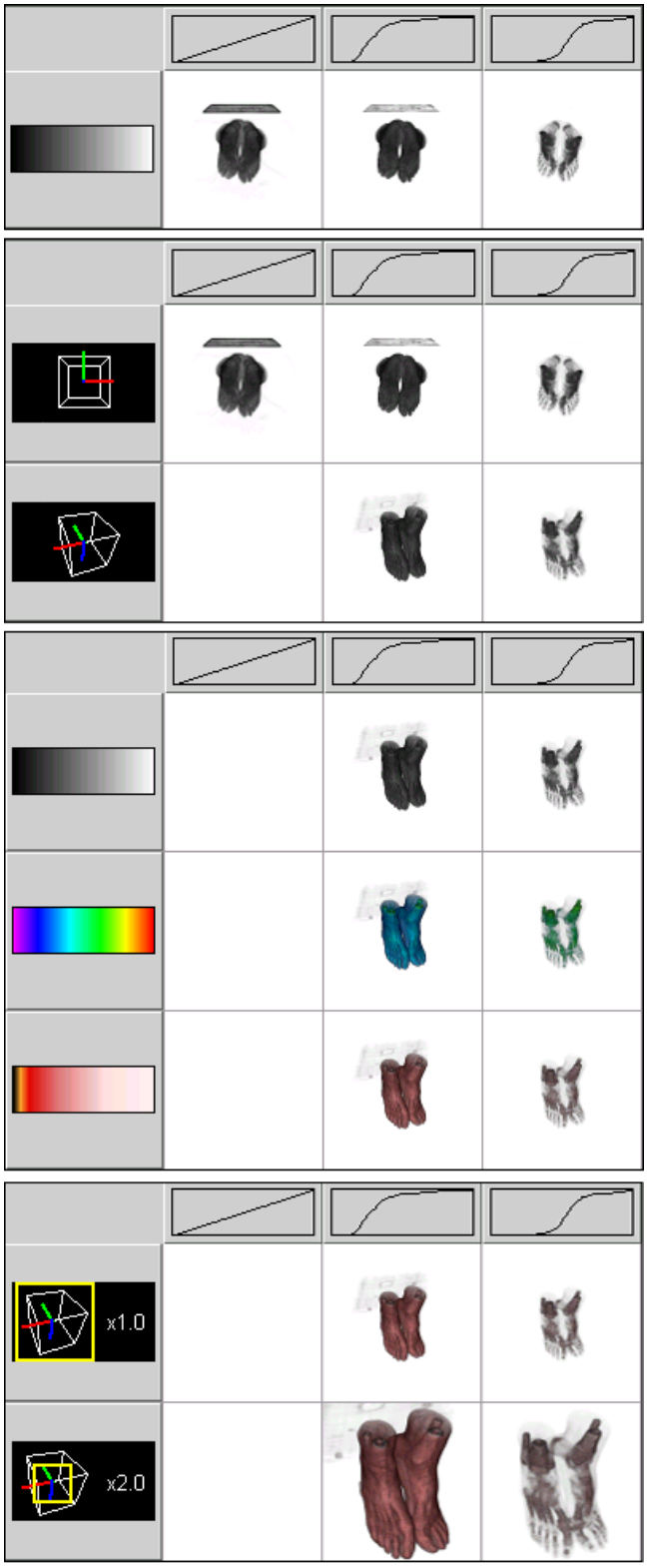

Figure 4: A sequence of spreadsheets displaying the visualization of a foot data set; blank cells represent non-rendered images. The goal was to compare skin and bone surfaces. The user first determined appropriate opacity maps before modifying the view position, color map and zoom factor. The spreadsheet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

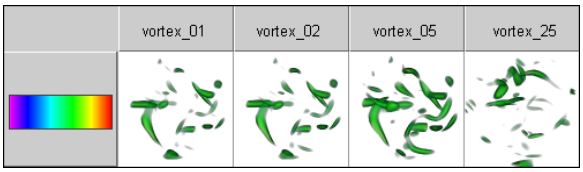

Figure 5: A spreadsheet displaying multiple data sets representing time steps in a vortex simulation. Modifying the displayed color map would change all the data set images at once, a task that would be more difficult in other representations.

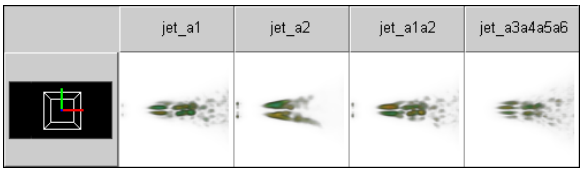

Figure 6: Another spreadsheet examining multiple data sets; the data represent distinct variables in a multi-variate turbulent jet simulation. The entire simulation has 9 variables; the first two columns are two of the variables. The third column is the sum of the first two variables over the entire 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.

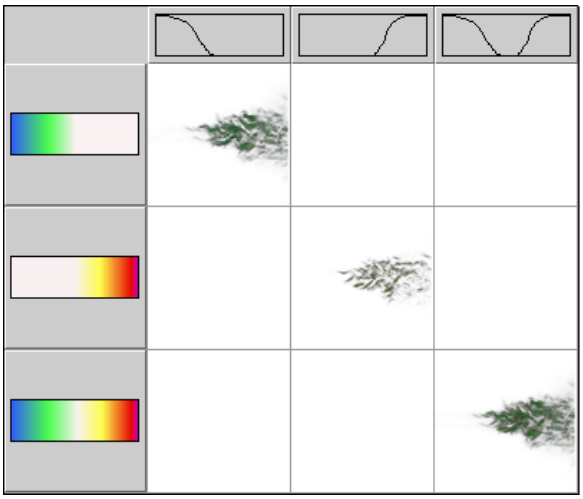

Figure 7: An example of applying parameter operators, in this case the union of color and opacity maps. The two original maps represent negative and positive vorticity, respectively, in a turbulent flow data set; the final image displays them together.

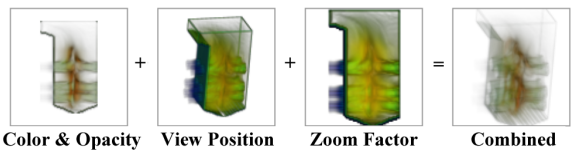

Figure 8: Selected cells for a composition operator; in this case, the user wishes to use the color and opacity maps of the first image, the view position of the second and the zoom factor of the third. The fourth image displays the desired visualization.