Title
Visualizing Visualization: A Model and Framework for Visualization Exploration

Author
Jankun-Kelly, T. J.

Publication Date
2003

Peer reviewed

Visualizing Visualization
A Model and Framework for Visualization Exploration

By

T J JANKUN-KELLY
B.S. (Harvey Mudd College) 1997
M.S. (University of California, Davis) 1999

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

_____

_____

_____

Committee in charge

2003

Visualizing Visualization
A Model and Framework for Visualization Exploration

The pyramid stones

A thousand generations

Each one given thanks

Acknowledgments

Too many people have assisted me over my 21 years of education to thank individually, and to leave any one out would lessen their importance. Thus, this work is dedicated to each pyramid stone that formed the foundation of this work.

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), ACI 9982251 (LSSDSV program) and 0222991, Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreements B347878 and B503159, Lawrence Berkeley National Laboratory, and the Director, Office of Science, of the U.S. Department of Energy under contract DE-AC03-76SF00098. Thanks go to Ayodeji Demuren, the Center for Computational Sciences and Engineering at Lawrence Berkeley National Laboratory, Philip Smith, Arthur Toga, Robert Wilson, and the Visible Human Project for the data sets used in this work.

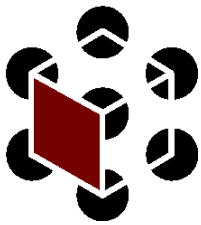
Contents

List of Figures

List of Tables

List of Symbols

Symbol	Description
G	Graph
v	Vertex
V	Set of vertices
e	Edge
E	Set of Edges
d	Data set
D	Set of data sets of a given type
P	Set of visualization parameter values of a given type
r	Visualization result value
R	Set of visualization result values of a given type
p	Parameter value set (p-set)
$p\,(i)$	Parameter value of type i in p-set p
P	Set of p-sets
$\mathcal{P}\,(A)$	Power set of the set A
t	Timestamp
t	Visualization transform
T	Set of visualization transforms
δ	Parameter calculus instance (derivation)
$\mathcal{D}$	Set of derivations
x	Parameter transform
X	Parameter transform list
s	Visualization session result
S	Set of visualization session results
$\mathcal{S}$	Visualization session
$\mathcal{V}$	Visualization space
ϵ	Data entity
$\mathcal{E}$	Set of data entities
m	Meta-data
M	Set of meta-data
$\rightharpoonup$	*next-result* relation; *next-results* relation
$\rightleftharpoons$	*same-timestamp* relation
$\leftharpoonup$	*previous-result* relation
$\Rightarrow$	*derives* relation
Δ_1	*differs-by-one* relation
l	String label



Visualizing Visualization

A Model and Framework for Visualization Exploration

Author: T.J. Jankun-Kelly
Committee: Dr. Kwan-Liu Ma, Chair
Dr. Michael Gertz
Dr. Kenneth I. Joy

Visualization and Graphics Research Group
Center for Image Processing and Integrated Computing
Department of Computer Science
University of California, Davis

Abstract

Visualization exploration is the process of extracting insight from data via interaction with visual depictions of that data. Visualization exploration is more than presentation; the interaction with both the data and its depiction is as important as the data and depiction itself. Previous visualization research has focused on the generation of visualizations—the depiction—and not on the exploratory aspects of the visualization process. However, without user interfaces for and formal models of the visualization process, visualization exploration sessions cannot be fully utilized. Towards this end, this dissertation introduces a model and framework for the visualization exploration process.

This research aims at providing a framework for capturing, representing, and manipulating information derived during the visualization discovery processes in a systematic manner. In particular, this work focuses on the exploration during the data analysis and visualization process through the use of intuitive graphical user interfaces. These interfaces provide a structured environment for the exploration of the visualization parameter space. The interfaces utilize a formal model of the visualization process that captures the fundamental operations performed during this exploration. The model is independent of the visualization performed or user interface utilized. In addition, instances of the model can be shared between users via an interoperable representation format. The goal of this work is to maximize user productivity by offering them an effective mechanism to fetch, edit, reuse, and share with others their visualizations which may include raw data, data associations, visualization results, and the steps taken to derive the visualization results.

Part I

Overview

Chapter 1

Introduction

The purpose of visualization is to extract insight from data through the use of images and animations of the data's features. The results of the visualization system are then used by the data creators or data users in their final analysis. Over the past decade, the field of visualization has matured; a wealth of techniques for a variety of data types have been developed to solve problems in various domains. As the use of visualization becomes more wide-spread, methods to support the use and dissemination of visualization must be developed. A visualization technique is of no use if there is no interface for that technique. If the results of that technique are not stored, then the technique is wasted. Thus, these problems must be addressed before visualization can become effective for large-scale deployment.

The following example illustrates the motivation for this research. Consider two scientists, Alice and Bob, who collaborate remotely. Bob is the primary data generator and Alice is the data investigator. Bob has generated a data set for Alice to visualize. Alice downloads the data and starts up her visualization system. Her system is a standard turn-key system—it allows the visualization parameters to be edited in iteration, but provides no access to previous results or parameter settings. At one point, Alice wishes to visually compare several different settings for a single parameter value. To do this task, she would have to manually re-input the previous parameter value since it is lost after subsequent editing; this redundant effort is costly in terms of the user's time and the computational

effort involved in regenerating the previous result. Alternatively, Alice could save each result externally of the visualization system, an unnecessary complication. When Alice's exploration is complete, she communicates her results to Bob. Unfortunately, since her visualization system does not store her visualization session, she must manually report her results and parameter settings to Bob. If Bob wants to further explore the data using Alice's results, he would have to enter these parameter settings manually. In addition, to share his results, he would have to repeat the laborious process of externally recording the images and parameter settings for Alice. As demonstrated, the lack of adequate data exploration support and session recording during the visualization process makes this form of collaborative work difficult.

1.1 Problem Statement

Current visualization systems do not structure the exploration process in order to eliminate costly redundant exploration. In addition, these interfaces do not provide context for their users. Once a visualization session is complete, many of these interfaces do not record the user's exploration. Thus, these explorations cannot be extended or shared with collaborators.

1.1.1 Approach

To solve these problems, this work describes a framework with the following components:

- An internal representation of the process.
- An external, visual representation of the process through user interfaces.
- A framework to communicate between the internal and external representations.

The user interface controls the visualization process; the session model records the visualization process. The internal representation consists of a model of the visualization process (based upon the fundamental operation of the visualization process described by this research) and a common representation of that process. This representation is understood

by the user interfaces using the developed framework. These user interfaces make use of four principles of visualization exploration that address the lack of exploration support in common visualization interfaces.

1.1.2 Contribution

The model and interface principles introduced here are powerful because they are general—they can be applied to a wide domain of visualization problems. The parameter space abstraction and process model can be used in many visualization tasks. The stored representation can be shared and extended in several different ways. This research will assist users of visualization in all disciplines to explore, communicate, and understand their results.

Chapter 2

A Characterization of the Visualization Process

This research aims to developing a model of the visualization exploration process and to describe principles for systems utilizing that model. In order to develop this model, the properties of the visualization process must be understood. This chapter examines different approaches to understanding the visualization process. The common feature of these approaches—iterative interaction during visualization exploration—will lead to a discussion of the properties of visualization systems that support this interaction. The properties of these systems will be distilled to illuminate the fundamental operation of the visualization exploration process. This fundamental operation characterizes the visualization process and is the basis for the visualization exploration process model discussed later.

2.1 The Purpose of Visualization

An oft quoted tenet of visualization is "the purpose of visualization is insight, not pictures."[1] Thus visualization is interdisciplinary by nature—the user of the visualization provides context for the visualization. By understanding visualization's place in the overall analysis process, a better understanding of the visualization process is gained.

[1] A quote derived from Richard Hamming's characterization of computation's purpose ("The purpose of computing is insight, not numbers," [25]) and paraphrased from McCormick et al. [42]

2.1.1 Scientific Visualization

The typical user of scientific visualization is an application scientist. In this setting, visualization is part of the scientific process. This process is based upon the assumption that physical phenomena can be described through objective means. To achieve objectivity, theories about the nature of phenomena should be testable and repeatable [44]. The scientific method was developed in order to satisfy the testability and repeatability criteria. The method consists of four steps: observation, theorization, experimentation, and analysis. Vital to the research discussed here are two properties of this process. First, the scientific method is inherently iterative: As experiments are analyzed, new observations could prompt new theories and experimentation. As a consequence, the scientific visualization process should also be iterative in order to support the hypothesize-test-analyze cycle of the scientific method. Secondly, documentation is vital to this process. Without documentation, other scientists cannot verify the experimental results, violating one of the central assumptions of the method. Consequently, visualization systems in scientific applications should document the visualization process in order to assist in repeatability. It should be noted that automated recording of the visualization can assist the scientist in another way—it can suggest patterns in exploration or new directions of possible study.

Springmeyer et al. [52] describe the entire scientific data analysis process of which scientific visualization is a part. This taxonomy was developed through interviews and observation of scientists. In their taxonomy, visualization is used mainly to interact with and maneuver through scientific data. The actions a scientist performs with visualization are summarized in the following list:

- Generating Data
- Examining Data
- Querying Data Values
- Data Navigation
- Data Value Comparison

- Data Value Classification

This taxonomy is important in that it provides a lists of requirements for scientific visualization systems. If a system does not support one of these actions, it is less useful to the scientist. When outlining principles for visualization interfaces in Chapter 8, this list will be revisited.

To summarize, visualization is utilized during the analysis portion of the scientific method. During this analysis, the scientist iteratively explores the data, using the results as documentation. As discussed next, users of information visualization utilize visualization in similar ways for similar reasons.

2.1.2 Information Visualization

Unlike scientific visualization, the users of information visualization systems are harder to encapsulate—they are not limited to scientists but include anyone using visualization in order to discover knowledge or make decisions. Put another way, the data sets of interest to users of information visualization are any sort of information that can be visually structured. This broad class of data is in contrast to the narrower traditional scientific visualization data sets—data sets which typically have a pre-determined physical structure. Since scientific visualization already has a structure, most research in that area has been upon increasing performance of various known visual mappings. In information visualization, more effort is given to creating informative mappings to create structure.

Human-computer interaction researchers [48] have studied how users integrate visualization into their work-cycle in a manner similar to Springmeyer et al. The "sensemaking cycle" (called "knowledge crystallization" in the more general context that does not use visualization [9]) models this task integration (see Figure 2.1). Sensemaking is the process of searching for a representation (a schema) and encoding data in that representation to answer task-specific questions utilizing a computer as an aid to cognition.

Information visualization can be used at every stage of the sensemaking cycle. If data is not available, information visualization and data mining can be used to extract data of interest. Visualization construction tools can be used to create an adequate visual schema,

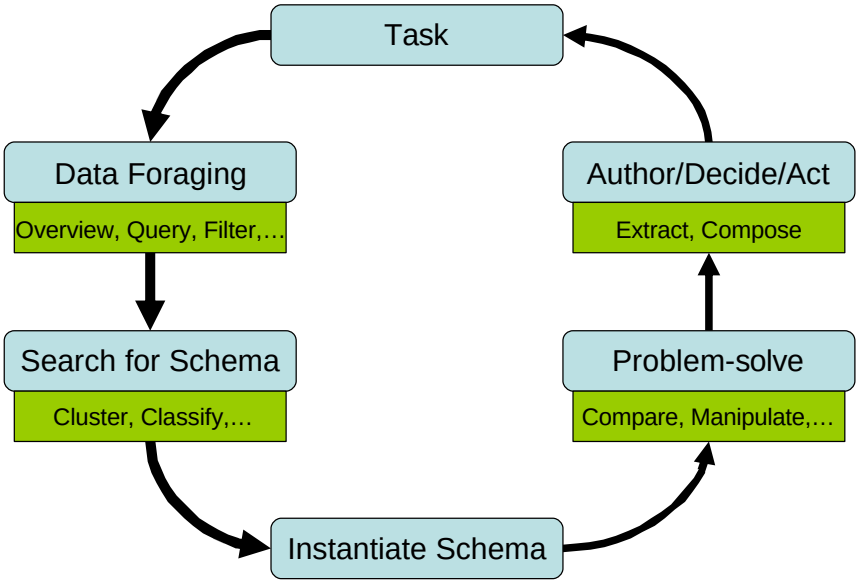


Figure 2.1: The knowledge crystallization cycle [9]; sensemaking is a form of knowledge crystallization that utilizes a computer to aid in cognition. Each stage in the cycle (blue) represents a major subtask, each of which contains different subtasks (green). At any stage in the cycle, the result of the task may cause the sensemaker to return to any previous stage.

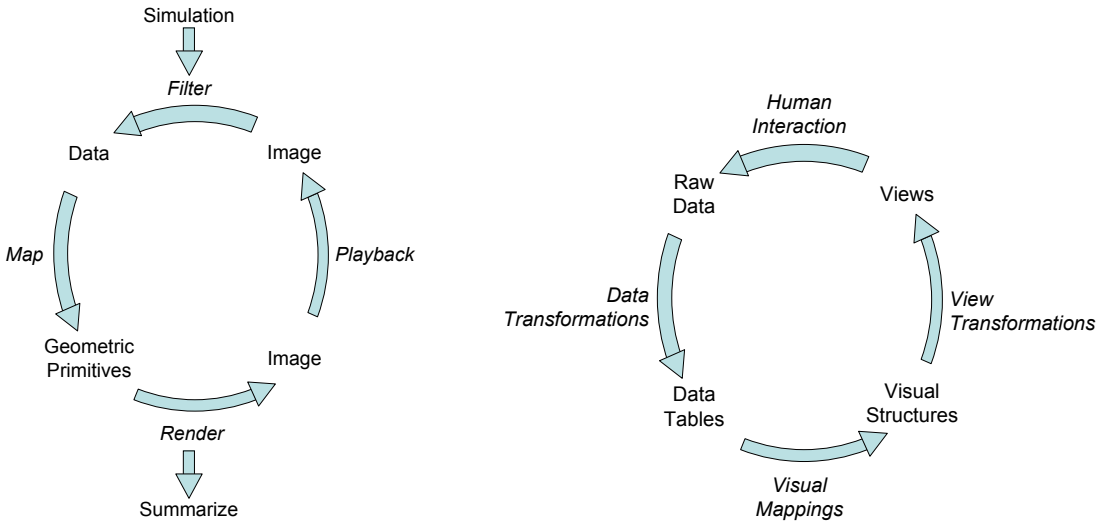
and then explore this schema to determine which features are vital and which are ancillary. Since sensemaking is cyclic, the visualization system could be used to iteratively explore as the data or the schema is changed. Thus, both information and scientific visualization systems must support iterative exploration.

2.2 The Visualization Exploration Process

During the visualization exploration process, a user interacts with a visualization system. The interaction leads to some sort of insight which is used in the overall task at hand. As discussed previously, the interaction with the visualization is iterative as new data or new insight could affect the exploration process. In order to model the visualization exploration process, a better understanding of this interaction is needed.

Upson et al. [56] describe the scientific visualization process as an analysis cycle. According to this model, data is filtered into subsets of interest, mapped onto visual primitives, and then rendered for the user by a function called the *visualization transform*. The

visualization generated by this transform is then used by the user to provide feedback into the previous steps, restarting the cycle. A similar cycle of raw data transformation, visual structure generation, and view rendering with user interaction is described by Card et al. [9] for information visualization. Figure 2.2 compares these two models. The key feature of both models is that the visualization process is an iterative sequence of user controlled transformations. Thus, elements that change during this iteration must be the focus of any description of the visualization process. These elements are arguments to the visualization transform—the visualization parameters. In Upson's model, these parameters control the data filtering process (e.g., by specifying a data threshold value), the visual primitive mapping (e.g., by changing colors assigned to data values), and the actual rendering (e.g., by changing lighting parameters). In Card's model, parameters manipulate the raw data transformation (e.g., by specifying what parts of a data table in a database to utilize), the visual mappings (e.g., by changing the mapping of the nominal variables from color to shape), and the view transformations (e.g., by changing the orientation of the view). As parameters are visualization transformation dependent, a description of this transform is also important to any documentation of the visualization process.



(a) Scientific visualization cycle [Upson] (b) Information visualization cycle [Card]

Figure 2.2: The visualization exploration cycle according for scientific visualization [56] and information visualization [9]. Both use a "filter-map-render" visualization pipeline controlled by user interaction.

While Upson and Card's models provide an overview for the visualization process, they are not fine grained enough to detail the user's exploration. What are needed are models that "unravel" the visualization cycle in order to discuss what the user was doing at any time in the exploration. Two approaches have been taken: visualization space paths and derivation models.

2.2.1 Visualization Space Paths

Several novel visualization user interfaces [39, 41] assume visualization exploration is equivalent to navigating a multidimensional visualization parameter space. In these models, each parameter in the visualization transform corresponds to a dimension in the visualization parameter space. Thus, a visualization result maps to a unique point in the visualization space. Users then trace a path through this space as they explore new results. For example, the Design Galleries [41] interface displays a overview of the entire space by random sampling while an Image Graph [39] follows a more structured path through the space. The models used by these interfaces provide an explicit tie between the visualization results (what the user sees) and the visualization parameters (what the user controls). However, the relationship between different results is not explicit in these model. Derivation models address this limitation.

2.2.2 Derivation Models

Derivation models describe how a new item was created from a previous item. For example, genealogy models the relationships between children and their parents (and previous generations). In visualization, two major derivation model efforts have been undertaken: The GRASPARC project [5] and Lee's general data exploration (GDE) model for visual data exploration [36, 37].

GRASPARC addresses modeling the search for a solution in a scientific problem solving environment (PSE). Like the visualization exploration process, this search is parameter driven; in this case, the parameters are the control variables for the simulation data in the PSE. A history tree structure is used to communicate and manipulate the PSE control state where nodes in the tree store the solution parameters and complete or partial results

(as simulations can be interrupted). A similar tree-like structure could be used for simple visualization process models; however, as discussed later, the types of interactions a user can have with a visualization system dictate that a more complex structure is needed.

Lee's work in visual database exploration provides the complex structure for process modeling lacking in the more simple GRASPARC model. Lee uses a graph-like structure to model the visualization process for databases. Vertices in the graph represent the state of the visualization while edges are relationships between states. These relationships are based upon similarities between meta-data attributes of the states and the data contained with the states. In Lee's work, the meta-data attributes describe structural attributes of the states. A suite of derivations are defined upon these attributes in order to capture the different interactions one can have with visual databases.

Both the visualization space and derivation models contain elements that are used in the proposed model for visualization exploration. The visualization parameter space provides a basis where visualization explorations can be embedded. The complex derivation model provide a formal definition of the relationships between different results in this visualization space. Instead of using structural meta-data attributes (as in the GDE model), derivations record the origin of the parameters used in a result. In other words, derivations describe how a user's interaction with the visualization system created the parameters that generate a visualization result.

2.3 Modeling User Interaction with Visualization

Visualization user interfaces allow the user to interact with visualizations. Thus, it is important to understand how a user interacts with such systems in order to model the visualization exploration process. Two approaches can be taken: A low-level, *syntactic* approach which examines how user interface events modify the visualization or a high-level, *semantic* approach which examines the goals of the user in interacting with the user interface elements.

Syntactic modeling of user interaction is the realm of human-computer interaction (HCI) research. This research generalizes to all forms of human-computer interfaces, not

just visualization interfaces. Hilbert and Redmiles [31] overview different models of user interface events and their effects. These models focus on interactions with the specific user interface element (i.e., "widgets" such as buttons or scrollbars) and how they correspond to different events in the software controlling the interface. Similarly, Duke et al. [18] discuss modeling the user interface elements themselves—they consider both the user and application interactions with the interface elements (called "interactors" in the paper). As these models do not address the intent of the user—the semantics of interaction—they are not the focus of the process model developed in this work.

Semantic modeling of visualization user interface interactions has been addressed in different ways. In information visualization, several automatic presentation systems—systems which create static visualizations of information with little or no human interaction—have investigated how the purpose of visualization affects the visualization's construction [10, 40, 47]. The GADGET systems [21, 22] extend these works to create automatic interactive visualizations based upon a user's intent; GADGET uses the Wehrend taxonomy of scientific visualization tasks [58] and Shneiderman's taxonomy of information visualization tasks [50] to determine the user's intent. These systems utilize a very high-level semantic model of visualization, one not tied to the user's interaction with the visualization. Thus, they are not sufficient for the research discussed here.

Chuah and Eick's Basic Visualization Interactions (BVI) [15] are another semantic model of user interaction for visualization. They decompose a visualization transform into atomic units of interaction—the BVIs. The BVIs include a visual control aspect (a user interface element) and methods for modifying the visualization. The user interacts with BVIs to control the visualization since the BVIs provide the interface for visualization parameter manipulation. This model reinforces the importance of visualization parameters to the visualization exploration process. However, Chuah and Eick's work does not address how the parameters can change—an important property for understanding the visualization process.

Rheigans [46] provides some insight into the different types of interactions a user can have with a visualization system. She discusses two types of interactions: interactive parameter control and dynamic manipulation. In the former, interactive manipulation of

the parameter values does not correspond to interactive updates to the rendered result; the result is only rendered upon user request. For dynamic manipulation interaction, parameter values can vary over a continuous range during manipulation. This range corresponds to a range of rendered results. Interactive parameter control interfaces were the norm in scientific visualization before the wide-spread availability of accelerated graphics hardware; dynamic manipulation is now common.

This classification will be used as the basis for the exploration process model discussed here. The only type of interactions not included in Rheigans' taxonomy are function derivations. In function derivations, parameter values are derived either by some sort of user-applied operator (as in the Image Graph [39]) or via a function created through an interactive interpreter (as in the VisSheet in Chapter 9). These three interactions capture all the interactions a user can have with parameters controlling a visualization in current visualizations. Chapter 5 discusses this model in more detail.

2.4 The Fundamental Operation of Visualization Exploration

From the previous discussion, several key properties of the visualization exploration process can be distilled. Visualization exploration is cyclic—parameters are modified iteratively until the results of interest are generated. The parameter values are generated in one of three ways: A parameter value can be generated from an old parameter value (through interactive parameter control); a range of parameter values can be generated from an old parameter value (through dynamic manipulation); or a set of parameter values can be derived from a different set of parameter values via some operator (through a function derivation). Parameter generation is part of the fundamental operation of visualization exploration:

> The *fundamental operation of the visualization exploration process* is the application of a set of parameter values to the visualization transform to generate a visualization result.

The fundamental operation defines the visualization exploration process. It also captures the key action a user performs during visualization—the generation of visualization results.

By describing how the user performs this fundamental operation, the entire visualization process is recorded. The framework discussed in this works facilitates the recording of the visualization exploration process and the exploration of the visualization space by assisting users in the fundamental operation.

Chapter 3

Overview of the Framework

The framework described in this research is designed to provide a structured environment for the exploration and dissemination of visualization results. The framework is divided into two main components—the internal and external representations of a visualization session (see Figure 3.1).

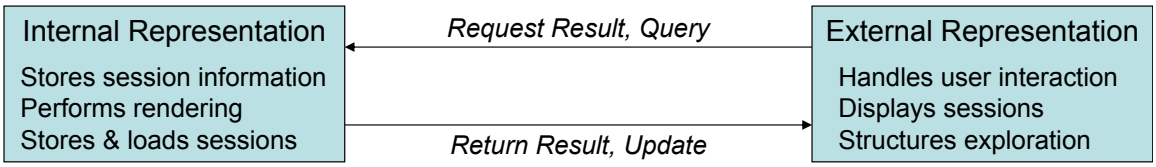


Figure 3.1: Overview of the Visualization Exploration Framework. The two components represent different parts of a visualization session: The internal representation stores the visualization results while the external representation is the interface to the visualization.

3.1 Internal Representation

The internal representation manages the visualization session for the user. When a new result is requested by the user (via the external representation), the internal representation generates the result, determines its relationships to previous results, stores the result and its relations, and returns the result to the user (via the external representation). In standard object-oriented language, it is the model in a model-view-controller (MVC) architecture [34]. All of the information stored by the internal representation can be accessed

by the user in some manner depending on the external representation. In addition, the internal representation can serialize the visualization session. The resulting serialized data can then be shared by other programs using the framework.

In this work, the internal representation is achieved by utilizing a formal model of the visualization exploration process. This model captures all the interactions a user can have with the visualization—these interactions were discussed previously during the characterization of the visualization process in Chapter 2. The definition of the model and its capabilities are discussed in Part II of this work.

3.2 External Representation

The external representation presents the user with the visualization. The visualization session is displayed to the user, allowing them to explore new results and compare them with previous results. In MVC terms, the external representation is both the View and the Controller. The external representation uses the internal representation in order to determine the state of the visualization process.

The external representation utilizes a set of principles of for visualization exploration discussed in Part III of this work. These principles give structure to the visualization exploration process, making it more efficient and effective. These principles will be demonstrated through a new spreadsheet-like interface called the VisSheet.

3.3 Summary

The Visualization Exploration Framework unifies the internal representation of a visualization exploration session with the external representation that depicts the session. The framework utilizes a formal model for visualization to capture the session and principles for visualization interfaces to modify the session. The interaction between the internal and external interfaces will be detailed in Part IV of this work.

Part II

Internal Representation

Chapter 4

Visualization Models

Before the formal model for the visualization exploration process is introduced, a better understanding of previous approaches to visualization modeling must be addressed. This work aims to describe the visualization process. To this end, it must detail the two major portions of the process: What visualization was done (i.e., the visualization transform), and how the visualization was done (i.e., the visualization session).

4.1 Visualization Transform Models

The visualization transform describes the type of visualization being performed. Formally, it is a function which maps data (the value) onto the graphical display (the view). Every type of visualization has an underlying visualization transform, though, in most cases, this transform cannot be changed as it is not apparent to the user.

A visualization transform model describes three things:

- The visualization transform (the mapping from data to result). There may be more than one visualization transform used during a single visualization session.
- The parameters which control the visualization transform (including the data set type).
- The result type generated by the visualization transform. Multiple result types may be needed if there are multiple transforms.

Each of the following models characterizes these three components differently (see Figure 4.1). However, their similarities signify that a high-level description of the transform, parameter types, and result types are sufficient to describe the type of visualization being performed. Low-level details utilizing one of the following models is only necessary in cases where the high-level description is not enough to determine the type of visualization being performed or when the visualization transform is to be modified using a supporting interface.

4.1.1 Data-Flow Model

The data-flow model, introduced by Haber and McNabb [24], is the transform model most widely used in scientific visualization applications. This model considers the visualization transform a pipeline where each stage in the pipeline represents a transformation. When these stages are collected, they form a network through which data flows. One reason the data-flow model is popular in scientific visualization is the number of data-flow interfaces for visualization [1, 35, 56, 61] which allow data-flow networks to be directly edited. For exploratory visualizations where the type of visualization desired is not known a priori, these interfaces are advantageous.

Formally, a data-flow visualization transformation network $G = (V, E)$ is a directed graph of visualization stages V connected by data-flow edges E. Each visualization stage $v_i = (I_i, O_i, f_i) \in V$ consists of a set of inputs I_i, outputs O_i, and a function $f_i : I_i \mapsto O_i$ which transforms the inputs into the outputs. Inputs are the parameters of the stage, while outputs are the results of the stage. Stages with no inputs and at least one output are parameters for the transform as a whole (i.e., network sources are transform parameters), while stages with at least one input are no output are the result of the transform (i.e., network sinks are transform results). The actual visualization transform $\mathtt{t} : V_{parameters} \mapsto V_{results}$ is performed by flowing the computation from the parameter vertices $V_{parameters} = \{v = (I, \emptyset) \in V : |I| > 0\}$ in the graph to the result vertices $V_{results} = \{v = (\emptyset, O) \in V : |O| > 0\}$ in the graph. If there is only one result vertex in the graph, the functions f_i performed at each stage v_i can be composed into a single function which represents the transform; if there are multiple outputs (e.g., a transform which

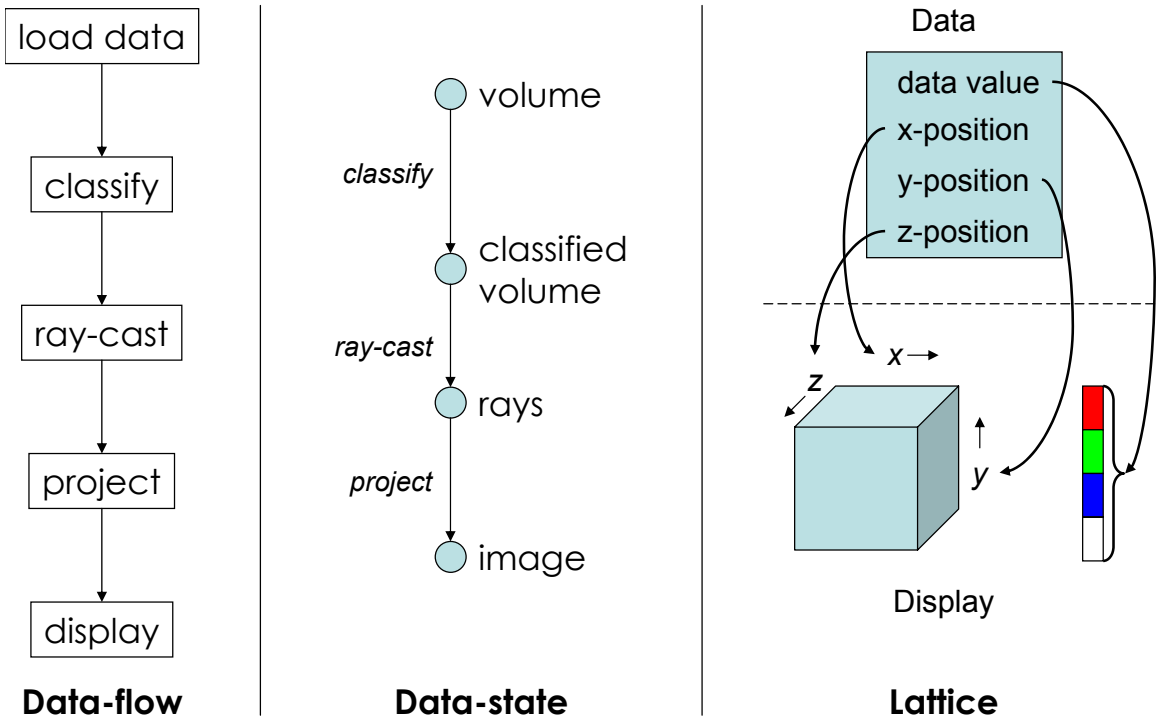


Figure 4.1: Characterizations of a ray-casting direct volume renderer in the data-flow, data state, and lattice visualization transform models.

produces two images for stereo output), each single path to a sink/result would produce a "sub-function" of the overall transform.

4.1.2 Data State Model

The data state model [14] focuses on the transformation of data states through the visualization pipeline. Like the data-flow model, a network is used to describe the visualization transformation. In a data state network, the nodes represent states of the data and the edges operations on the data. Operators are classified according to the stage in the information visualization pipeline in which they occur (Figure 2.2); for example, the classification stage of a volume renderer is a data transformation operator since it changes an unclassified volume into a classified volume. Though the data state model can be used to describe scientific visualizations, it has been mostly applied to descriptions of information visualizations [12]. At the time of this writing, there were no interfaces to edit or present data state networks in a similar fashion to data-flow networks.

Formally, a data state visualization transform network $G = (V, E)$ is an directed graph of data states V connected by operation edges E. Each data state $v_i \in V$ is a representable piece of data. Each edge $e_i = (v_{from}, v_{to}, f_i) \in E$ performs a visualization operation $o_i : v_{from} \mapsto v_{to}$ which transforms the source vertex (its input parameter) into the destination vertex (its output result). As formulated, the data stage model is ambiguous when determining parameters and results of the transform, since operators with more than one input or output are not explicitly addressed. Under simplifying assumptions (single-input, single-output operators), it has been shown that similar data-flow and data state networks are the dual of one another [11]. In this case, the visualization transform $\mathsf{t} : v_0 \mapsto v_n$ is the composition of all the data operators o_i where v_0 is the vertex with no in-going edges from other vertices (the input) and v_n is the vertex with no outgoing edges to other vertices (the result).

4.1.3 Lattice Model

The display model used by VisAD [29, 30] is interesting in that it maps data attributes directly to display via mappings utilizing lattice theory and visual presentation principles developed by Bertin [2] and Mackinlay [40]. Lattices of data and displays are used, where a lattice provides a partial ordering of the accuracy of the data and display (from various meta-data attributes). The visualization occurs by using a function to map between the lattices. The mapping is determined by maximizing the expressiveness of the display, where expressiveness is defined by Mackinlay: A display expresses a fact if the display encodes all those facts and only those facts [40]. Because the model describes both the data and the display, it is both a visualization transform model (as it describes the mapping of value to view) and a visualization data model (as it describes the structure of the data). As of this witting, VisAD [28] is the only system utilizing this model.

Formally, this model uses two lattices U and Y to represent the data and display structure respectively. U represents the lattice of possible values and types in the data (each of the with an order relations based upon accuracy) while Y is a lattice of tuples describing graphical marks (positions in a 3D-display, a time for the display, and a 3-tuple for the red, green, and blue colors of the mark). The construction of these lattices

are beyond the scope of this discussion. To satisfy the expressive criteria, the display function $f : U \mapsto Y$ must be a lattice isomorphism between U and a sub-lattice Y' of Y (i.e., if the ordering relationship is written as $\vee$, then $f(u_1 \vee u_2) = f(u_1) \vee f(u_2)$ where $u_1, u_2 \in U$). Y' is the sub-lattice that does not contain Y's maximal element. The display is constructed by creating a pair of functions $g, h : \mathbf{R} \mapsto \mathbf{R}$ such that for each type within U, $f(\{\perp, \ldots, \perp, [x, y], \perp, \ldots, \perp\}) = \{\perp, \ldots, \perp, [g(x), h(x)], \perp, \ldots, \perp\}$ where $\perp$ signifies the undefined element (remember that each element in the tuple corresponds to a simple type in the data lattice). Thus, each type P in the lattice is expressed by exactly one display variable in the display. Therefore, to construct the visualization transform $\mathtt{t} : P \mapsto Y'$, each type in the display lattice acts as a parameter in the visualization.

4.1.4 Evaluation

	Describes					
Name	Transform	Parameters	Results	Editable	Composable	Formalism
Data-flow	Yes	Yes	Yes	Yes	Yes	Good
Data State	Yes	Yes	Yes	No	Yes	Ambiguous
Lattice	Yes	Yes	Yes	Yes	No	Strong

Table 4.1: Summarization of the visualization transform models.

Each of these models have different strengths and weaknesses. The data-flow and data state approaches are very similar, and, as mentioned, are duals of each other in certain circumstances. Data-flow benefits from being better studied due to its longevity and the availability of data-flow network editing interfaces. Neither of these applies to the data state model. The data state model also suffers from its ambiguity in definition. However, the data state model better captures information visualization transforms since data-flow networks have not been traditionally used in that area. In addition, the state of the data is explicit in the data state model, whereas this information must be gathered from context in the data-flow model.

The data-flow and data state models are high-level models whereas the lattice model is low-level. The lattice model provides a solid formal description of the visualization

transform at the expense of providing a succinct overview of that transform. In addition, it is unclear how composable operations such cropping or display operations such as composition are handled using the lattice model.

This work will utilize an expanded data state model that disambiguates the original data state model and clarifies the role of the parameters in the transformation. This model was chosen as it can be utilized to describe both scientific and information visualization transforms. In the future, to provide a more formal foundation, it would be interesting to peruse building the model out of the lattice formalization. Perhaps a composition of lattices describing each stage in the data state pipeline could be utilized.

4.2 Visualization Data Models

To fully formalize the visualization transform model, the types of the data sets, transform parameters and transform results must be known. For documentation purposes, a simple label denoting the type may be sufficient. For transform results, the label may be a Multipurpose Internet Mail Extension (MIME) media type [20] describing the output format. For data sets, it might identify the physical data format used to store the data. However, to formally describe the data, parameters, and results generated during the visualization process, a more robust data model is needed. Data models for scientific visualization data sets [7, 23, 30, 55] and information visualization data sets [8, 9, 50] have been developed, but may not be applicable to visualization transform parameters and results. More research unifying scientific and information visualization data models needs to be done. For this work, unique names for the data, parameters, and results types are utilized (MIME type names are used for results) until a more general data model can be developed.

4.3 Visualization Session Models

The visualization session model describes what occurred during a visualization session. At the least, it records the results of the user's process. Most visualization user interfaces only provide limited interaction with the stored visualization process. For exam-

ple, a simple undo/redo of parameter editing could be supported or a display of previous results.

A complete visualization exploration session model describes four components:

- The visualization results generated during the visualization process.
- The parameter values used during the process.
- A linear history of the generated results.
- An encoding of the relationships between results.

None of the models discussed during the characterization of the visualization process captures all four elements, though the GDE model does possess information that could be utilized for a similar purpose. The process model formalized by this work does capture these four elements.

4.3.1 Visualization Space Path Model

The visualization space path model used in several interfaces [39, 41] posits that visualization exploration is the process of exploring a path through visualization parameter space. Each visualization result represents a unique point in this visualization space. Depending on the interface, different visualization paths can be used to explore this space.

Formally, given a visualization transform $\mathtt{t} : P_0 \times \cdots \times P_n \mapsto R$ mapping parameters $P_0, \ldots, P_n$ to results R, a visualization parameter space $\mathcal{V}$ is an n-dimensional space, one dimension for each parameter. To be a proper space, an arbitrary ordering can be assigned to parameter types without a natural ordering (this work never utilizes this ordering criterion). Each point in the space, identified by a set of parameter values (a *p-set*), corresponds to a unique result $r \in R$—if $p \in P_0 \times \cdots \times P_n$ is a p-set, then $r = \mathtt{t}(p)$. A visualization path is a sequence of result sets $S = (S_0, S_1, S_2, \ldots)$ where each $S_i \in \mathcal{P}(R) - \emptyset$ ($\mathcal{P}(A)$ is the power set of set A—the set of all subsets of A). Each S_i represents all the results generated during time step i in the exploration. For example, in the Design Galleries system, S_0 consists of a large set of initial results (generated from the random sampling) with each following S_i consisting of a single derived result. In most visualization systems, a visualization space

path would only consist of sets of one element as only one result can be displayed and edited at a time.

4.3.2 GRASPARC Model

The GRASPARC system [5] utilizes a tree of PSE parameters to model the exploration process. If used to model visualization exploration, elements in the tree become p-sets. Derivation information is encoded in the tree. Formally, given a visualization transform $\mathtt{t} : P_0 \times \cdots \times P_n \mapsto R$, the process tree $G = (R, E)$ defines the exploration where the edges $E \subset R \times R$ are such that $(r_i, r_j) \in E$ if r_i is r_j's parent. This tree can model interfaces where a user can return to any previous result in the exploration to start a new branch of the exploration.

4.3.3 General Data Exploration Model

Lee's General Data Exploration Model (the GDE Model [37]) is built around data entities, meta-data describing the data, and data derivations. Data entities consist of the underlying data, the meta-data annotating the data, and a label for the entity. In addition, the temporal ordering of the derivations can be visualized using derivation sequences while data derivation and lineage graphs display data relationships. Derivations in the GDE Model describe which data entities were used to generate subsequent data entities. In addition, a time stamp and meta-data describing the derivation are also associated with a derivation.

Formally, the GDE Model consists of data entities derived using data derivations[1]. A data entity $\epsilon = (d, k, M)$ consists of a data set d (described using some data model), a key k, and meta-data $M = (m_I, m_S, m_P, m_K)$ describing the identification (m_I), structural (m_S), process (m_P), and knowledge (m_K) attributes of the data d. A data derivation $\delta = (f, t, M)$ consists of a binary relation $f : \mathcal{E}_i \mapsto \mathcal{E}_o$ between an input set of data entities $\mathcal{E}_i$ and an output set of data entries $\mathcal{E}_o$ where $\mathcal{E}_i \cap \mathcal{E}_o = \emptyset$, a time stamp t, and meta-data m describing the derivation. A derivation sequence is then an sequence $\mathcal{D} = (\delta_0, \delta_1, \ldots, \delta_n)$

[1] In Lee's formalism, there are three different classes of data entities and data derivations based upon the precision of the meta-data associated with each; for the purpose of this discussion, the distinctions are not important.

of data derivations $\delta_i = (f_i, i, m_i)$—a derivation sequence fulfills the same purpose as a visualization path. Similarly, a data derivation graph $G = (V, E)$ details the relationships between data entities where, given a derivation sequence $\mathcal{D} = (\delta_0, \ldots, \delta_n)$, the vertices $V = \bigcup_i \text{domain}(f_i) \cup \bigcup_i \text{range}(f_i)$ (where $f_i \in \delta_i$ is δ_i's binary relation) consist of all the data entities in the domain or range of the any data derivation and a directed edge exists between two vertices v_i and v_j if and only if $f_j(v_i) = v_j$ for some $f_j \in \delta_j \in \mathcal{D}$.

4.3.4 Evaluation

	Describes				
Name	Results	Parameters	History	Derivations	Formalism
Visualization Space Path	Yes	Yes	Yes	No	Good
GRASPARC Tree	No	Yes	No	Partial	None
GDE Model	Yes	No	Yes	Yes	Strong

Table 4.2: Summarization of the visualization session models.

Of the considered process models, the GDE Model is the most complete, fulfilling three of the four criteria for a visualization session model. Visualization space models provide a good space in which the visualization process could be embedded and thus tracks which parameters generated a result. However, these models do not encode how a result was generated. The GRASPARC tree indicates which result was generated from another result but does not specify how that result was generated from its parent result nor when that result was generated. In addition, it is implicit in the GRASPARC model that only one result can be generated from a previous result in any time step. Neither of these limitations apply to the GDE Model, since a data derivation consists of an arbitrary binary relationship with a time stamp. However, the GDE Model does not specifically formulate the parameters used to create a visualization result nor how those parameters were generated (some of this information may or may not be in the meta-data).

In the visualization session model developed here, an extension of the visualization path models and the GDE Model is used. The model utilizes the visualization parameter space from the path models as the framework for the model. Parameter value sets take the

place of data sets and visualization session results containing p-sets, results, time stamps, and derivation information take the place of data entities. Unlike the GDE Model, derivation information is embedded directly in the session results. Sequences of sessions results give a temporal ordering of the derivation and visualization derivation graphs can also be generated from the session results. Thus, the four properties of a session model are satisfied.

4.4 Summary

A visualization exploration process model consists of a visualization transform model and a visualization session model. Previous approaches to visualization transform modeling have mostly addressed the three requirements for such models, and thus an entirely new transform model does not need to be introduced. However, previous session models have been limited in their ability to express either different kinds of derivations or the parameters involved in those derivations. Thus, the major contribution of the visualization exploration process model in this work is the session model. The details of this model are provided next.

Chapter 5

A Visualization Exploration Process Model

This chapter introduces the P-Set Model for Visualization Exploration (the P-Set Model for short). It formally describes the visualization exploration process by describing how the visualization was performed (using a visualization transform model), how user interaction with the visualization generated new results (using a new parameter derivation calculus), and how a sequence of these interactions are related (using a new visualization session model). The model is illustrated through several examples that demonstrate its capabilities.

5.1 Visualization Transformation Model

The visualization transform model outlines the type of visualization being applied. Without this information, it may be difficult to determine how the visualization results were generated. The transform also details what parameter types are used to generate a result. This information is needed by the session model in the next section.

The visualization transform model described here augments the data state model to include information about the parameters used in the visualization process. The data state model was chosen because it already focuses upon the data sets utilized in the visualization, an important parameter. Formally, the transform model consists of the following

components:

Visualization transform A function $\mathtt{t} : D_0 \times \cdots \times D_n \times P_0 \times \cdots \times P_m \rightarrow R$ which describes the mapping of value (the data set types D_0 through D_n) to view (the visualization transform result type R). P_0 to P_m are *visualization transform parameter types*.

Visualization transform parameter type Any set that is part of the domain of the visualization transform function. By this definition, data set types are also visualization transform parameter types. A member of a visualization transform parameter set is a *visualization transform parameter value*, or parameter value for short.

Visualization transform result type A set which is in the range of a visualization transform function. Members of this set, known as *visualization transform result values*, or results for short, are directly representable in graphical form (such as a raster image, shaded geometry, etc.).

When documenting the visualization exploration session, only the visualization transforms and corresponding parameter and result types used need to be recorded. This can be accomplished by listing the parameter types and results type for each transform (called the transform signature). For a more detailed breakdown of the transform, the full formal model can be used.

The complete visualization transform model decomposes the visualization transform into stages of visualization operator applications. A visualization operator $o_i : D_{i,0} \times \cdots \times D_{i,n} \times P_{i,0} \times \cdots \times P_{i,m} \mapsto D_{i,n+1}$ maps a set of data elements ($D_{i,0}$ through $D_{i,n}$) and operator parameters ($P_{i,0}$ to $P_{i,m}$) to an output set ($D_{i,n+1}$). Visualization operators are composed in order to form a visualization transform. Though data sets are considered a type of parameter, the model does force a distinction—parameters are controllable values exposed by the user interface while data sets are provided by the user. The former provides control over the operation while the later is the input to or the output of the operator.

As stated, a visualization transform is the composition of a sequence of visualization operators: $\mathtt{t} = o_j \circ o_{j-1} \circ \cdots \circ o_0$. The output data set of any operator becomes one of the input data sets for the subsequent operator (e.g., output data set $D_{i,n+1}$ for operator

o_i becomes $D_{i+1,0}$ in operator o_{i+1}). To determine the signature of `t`, all the input data sets to any operator which are not the output of a previous operator are considered the visualization data sets while all of the parameters to the various visualization operators are considered the visualization parameters of `t`. Finally, the output data set of the last visualization operator is considered the visualization result. Note that the same parameter type may be used in different visualization operators (i.e., $P_{i,s}$ may have the same type as some $P_{j,t}$); they may or may not share the same parameter value.

Like the original data state model, a decomposed visualization transform in the expanded model may also be graphically displayed. For the graph $G = (V, E)$, each vertex represents one of the data set types (a data state $D_{i,s}$ for some i and s). There is a directed edge $e_i = (V_{from} \subset V, v_{to} \in V, o_i) \in E$ between vertices $V_{from} = (v_0, \ldots, v_n)$ and v_{to} if the corresponding data stages for the $v_i \in V_{from}$ are in the domain of o_i. This graph distinguishes between a vertex with multiple incident edges and a vertex with a single edge connected to multiple incident vertices. In the former, several visualization operators map to the same data state. In the later, several data sets are used to derive a single data state. This property disambiguates the original data state model. Similarly, the expanded definition of the visualization operator function includes the parameter types explicitly, a property missing from the original specification.

5.2 Visualization Session Model

The visualization session model serves two purposes. Its primary purpose is to capture the path of exploration during the visualization session. This information encapsulates the details of the visualization exploration process. The secondary purpose of the model is to allow the description of the session to be further analyzed, visualized, or manipulated.

Unlike the other visualization session models, this model describes four components: the results, the parameter values, the history of the results, and the result's relationships. The visualization results are recorded because they are the desired outcome of the user's exploration process. Each result is uniquely identified by the parameter values which generated that result. Without the parameter values, the final results could not be

reproduced. Finally, the history and derivation information is needed to reproduce the visualization process.

As previously stated, the fundamental operation that occurs during the visualization process is the formation of parameter value sets to derive visualization results. These parameter value sets, or p-sets, posses a parameter value for each parameter in the visualization transform. New p-sets are created by user interaction with the visualization system. The session model tracks the relationships between results by recording how a user generates new p-sets (and thus results) from old p-sets. P-set derivations can be expressed as one of three templates using a *parameter derivation calculus*. In the following, the $p_j = \{p_j(1), \ldots, p_j(n)\}$ are p-sets and each $p_j(i) \in P_i$ is a different parameter value for the same parameter type P_i:

I. *Parameter Application* $p_2(i) | p_0 \mapsto p_1$: parameter value $p_0(i) \in p_0$ is replaced by $p_2(i) \in p_2$ in order to derive p-set p_1.

II. *Parameter Range* $[p_0(i), p_1(i)] | p_0 \mapsto p_1$: a continuous range of parameter values is generated between discrete parameter values $p_0(i)$ and $p_1(i)$ and applied to p-set p_0. p_1 represents the p-set at the end of the continuous interaction.

III. *Function Application* $p_0(i) \rightarrow p_1(i) | p_0 \mapsto p_1$: parameter value $p_1(i)$ was calculated from $p_0(i)$ by some function and then applied to p_0 to generate p_1.

All derivations are expressed as *parameter-transform-list* | *input-tuple* $\mapsto$ *output-tuple*. Such constructions are *parameter derivation calculus instances*. The parameter transform list contains input and output parameter values; the former is used to derive the later. Output parameter values are sequentially applied to the elements in the input p-set tuple to generate the output p-set tuple. In the templates, $p_0(i)$ is an input parameter value while $p_1(i)$ and $p_2(i)$ are output parameter values. It is possible to have multiple input parameter values, output parameter values, p-sets in an input tuple, or p-sets in an output tuple. For example, Figure 5.1 demonstrates a function derivation with two output parameter values (in braces) and two elements in the output tuple (in angle brackets).

Formally, a parameter calculus instance $\delta = (X, \mathsf{P}_{input}, \mathsf{P}_{output})$ consists of an sequence of input and output p-sets P_{input} and P_{output} respectively and a list of parameter

transformations $X = (x_0, \ldots, x_n)$, $x_i = (l_0, (p_i(j), \ldots), (p_{i'}(j'), \ldots))$, consisting of a label l_i determining what transformation occurred (parameter application, parameter range, or function application), a list of input parameter values, and a list of output parameter values for that transformation. Derivations occur as follow: All possible parameter combinations which form valid sub-p-sets derivable from the output parameter values are generated (a sub-p-set is a set of parameter values which may not include all the parameter types specified by the visualization transform). These are then applied in order to all the input p-sets in P_{input} to create P_{output}.

The parameter derivation calculus describes all the salient parameter behaviors described in Chapter 2. The calculus is the basis for recording the visualization exploration session. Formally, a visualization session consists of a set of *visualization session results*. A visualization session result $s = (p, r, t, \delta)$ is a tuple containing a p-set p, the visualization transform result derived from the p-set r, a timestamp t to place the result in temporal context, and a parameter derivation calculus instance δ detailing how the result was derived. Example session results are given in Figure 5.1. Each session result represents the generation of a single visualization result. As the example illustrates, it is possible for parameter calculus instances to be the same for two or more session results (the third and fourth lines in the example). This disparity is due to the fact that calculus instances correspond to user actions while session results correspond to rendered results. The same user action can create more than one rendered result, all sharing the same timestamp. Though it is possible for a user to re-visit the same visualization result by generating the same p-set more than once, each is a unique session result identified by a distinct timestamp. A complete visualization session $\mathcal{S} = (T, \mathsf{P}, R, S)$ then consists of the set of transforms T used in that session, the p-sets P used in the session, the visualization results R generated from the p-sets, and the corresponding visualization session results S.

One issue remaining is the change of visualization transforms during a visualization exploration session. Changes of visualization transform are not explicitly encoded in the model. Currently, it is assumed that visualization transforms can be uniquely identified by their signature: the parameter types and result type used in the visualization transform. Thus, a p-set not only uniquely identifies a visualization result but also identifies the visu-

$$
\begin{array}{l}
s_0 = \langle p_0, r_0, t_0, \emptyset \rangle \\
s_1 = \langle p_1, r_1, t_1, p_1(2) | \, p_0 \mapsto p_1 \rangle \\
s_2 = \langle p_2, r_2, t_2, p_1(1) \rightarrow \{p_2(1), p_3(1)\} | \, p_1 \mapsto \langle p_2, p_3 \rangle \rangle \\
s_3 = \langle p_3, r_3, t_2, p_1(1) \rightarrow \{p_2(1), p_3(1)\} | \, p_1 \mapsto \langle p_2, p_3 \rangle \rangle \\
s_4 = \langle p_4, r_4, t_3, p_4(2) | \, p_0 \mapsto p_4 \rangle
\end{array}
$$

Figure 5.1: A series of visualization session results. A session result is a tuple of a p-set (the p_i), the visualization result corresponding to that p-set (r_i), a timestamp (t_i), and information detailing how the result was derived. In this example, the second session result was derived from the first in the second timestep before the third and fourth results were both derived in the third timestep. Afterwards, the fifth result was derived from the first.

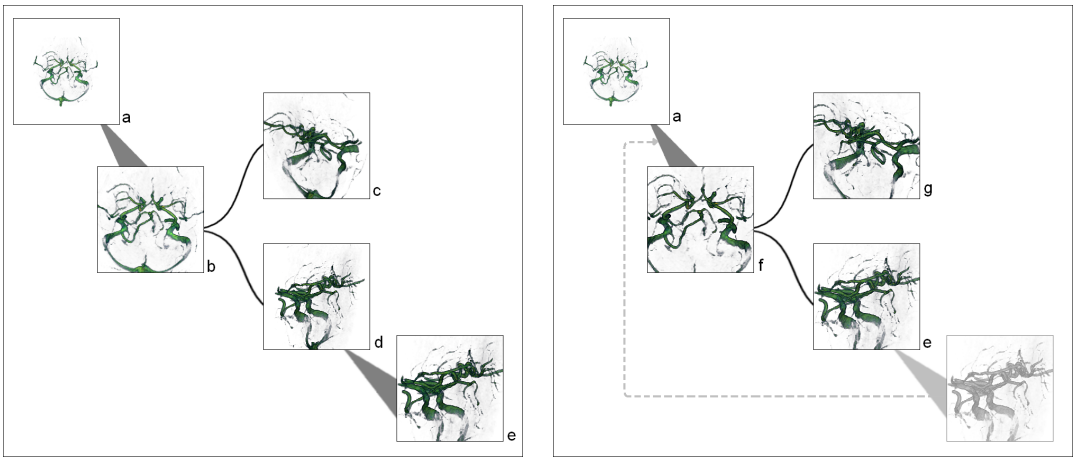
alization transform that generated the result since the parameter types are defined by the transform. Explicit identification of a transform change is not needed since it is implicit in the visualization session result sequence.

5.3 Examples

To better understand the details of the visualization exploration process model, we present a few examples. Three examples are presented in this section. The first two examples consider detailed visualization sessions in order to demonstrate the effectiveness of the model and representation. They also provide concrete examples of how the parameter calculus can describe real visualization sessions. The first uses an Image Graph user interface while the second one uses a spreadsheet-like interface. The last example is a case study demonstrating how an implementation of the model was added to an existing visualization tool.

5.3.1 Image Graph Example

The first example (Figure 5.2) demonstrates the use of the model and representation. A volume visualization of blood vessels in the brain was performed using the Image Graph (top left image in the figure). The user first zoomed into a region of interest (result s_b). Two rotations were then used to display different views of the vessel (s_c and s_d). After zooming in again (s_e), the user decided to apply the final zoom magnification to the earlier images. This was accomplished by dragging the zoom edge over the previous zoom edge.



$s_a = \langle p_a, r_a, t_0, \emptyset \rangle$	Default
$s_b = \langle p_b, r_b, t_1, p_b(zoom) \mid p_a \mapsto p_b \rangle$	Zoom a
$s_c = \langle p_c, r_c, t_2, p_c(view) \mid p_b \mapsto p_c \rangle$	Rotate b
$s_d = \langle p_d, r_d, t_3, p_d(view) \mid p_b \mapsto p_d \rangle$	Rotate b
$s_e = \langle p_e, r_e, t_4, p_e(zoom) \mid p_d \mapsto p_e \rangle$	Zoom d
$s_f = \langle p_f, r_f, t_5, p_e(zoom) \mid \langle p_b, p_c, p_d \rangle \mapsto \langle p_f, p_g, p_e \rangle \rangle$	Zoom b
$s_g = \langle p_g, r_g, t_5, p_e(zoom) \mid \langle p_b, p_c, p_d \rangle \mapsto \langle p_f, p_g, p_e \rangle \rangle$	Zoom c
$s_{e'} = \langle p_e, r_e, t_5, p_e(zoom) \mid \langle p_b, p_c, p_d \rangle \mapsto \langle p_f, p_g, p_e \rangle \rangle$	Zoom d

Figure 5.2: Representation of a brain vessel visualization. The feature of interest is the bulge in the lowest vessel in image e (top left image, captions added for clarity). During the visualization, the user dragged the zoom edge going to image e over the edge from a to zoom the other images in the Image Graph (top right). These derivations, including the propagation of the zoom factor from e to results f and g, are recorded in the session results (bottom).

The images using the new magnification (from $s_{e'}$, s_f, and s_g) replaced the old images (from s_d, s_b, and s_c respectively) to produce the image graph shown in the top right image in the figure. During the exploration, the session results were recorded (bottom portion of the figure); these results explicitly state how the zoom parameter value from s_e's p-set was applied to results s_b and s_c to derive results s_f and s_g respectively (the fifth and sixth lines in the bottom portion of the figure). Due to the propagation, the same p-set p_e was generated twice when p_e's zoom factor was applied to p_d, p_e's parent. When this session is analyzed in Chapter 6, the distinction between session results and p-sets will become more apparent.

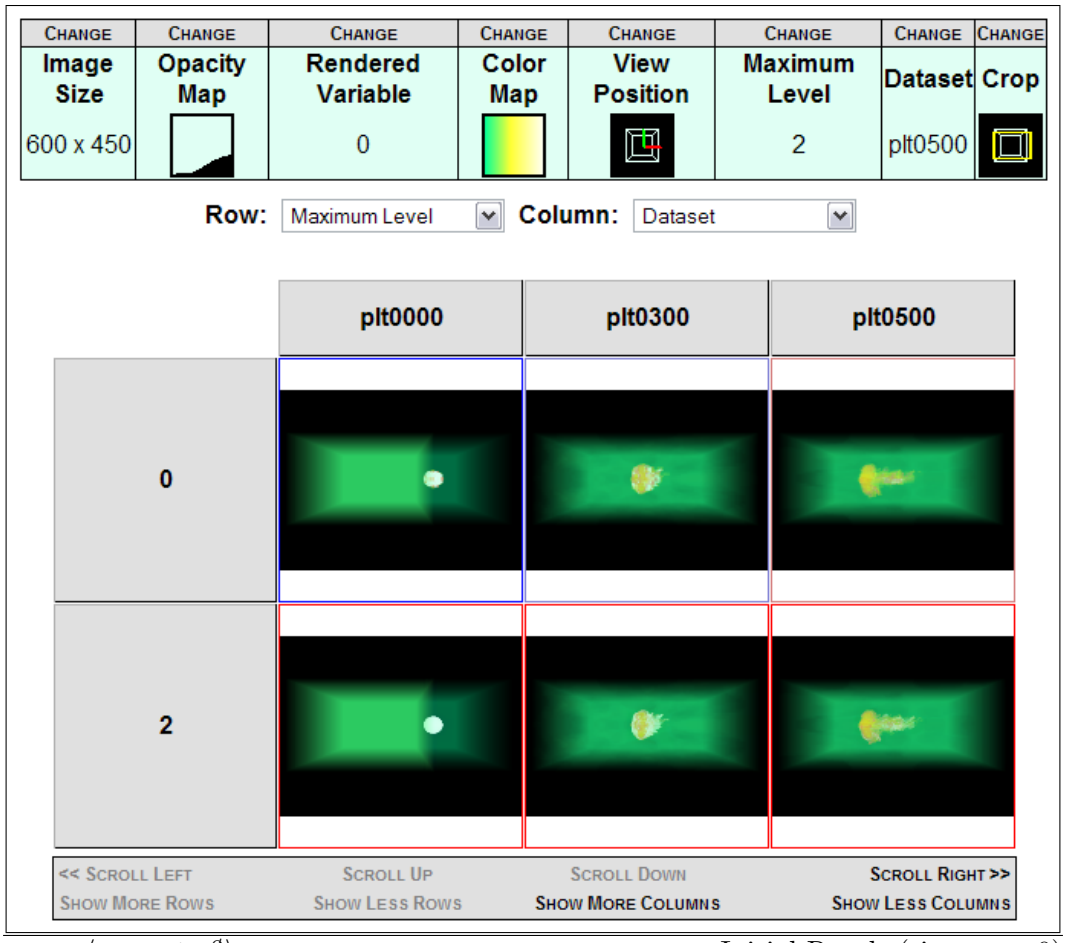
5.3.2 VisSheet Example

In this example, the session consists of results from a visualization of a multi-resolution, time-varying computational fluid dynamics (CFD) dataset exploration (Figure 5.3). A web-based spreadsheet-like interface (the WebSheet, see Chapter 9 and 10) was used. In this example, the user first generated three results at different simulation time-steps at the coarsest resolution (results s_0–s_2). The user then simultaneously requested the same three images at the highest resolution by adding a new row (results s_3–s_5); the final state of the interface can be seen at the top of Figure 5.3. Since last three results were generated at the same time, they share the same timestamp t_3. The session results completely capture this visualization exploration session.

5.3.3 Dynamic Manipulation Interface Example

In the last example, the model was used to augment an existing visualization tool. The tool in this example is used to visualize anomalies in Internet routing using the Border Gateway Protocol (BGP) [54]. The tool displays different types of changes to ownership of autonomous systems (ASes)—groups of hosts on the Internet. The different types of changes correspond to different colors (top left image in Figure 5.4). Lines of the appropriate color connect an AS along the edge of the square to IP addresses affected by the AS change within the square. The tool allows a user to browse through different dates with different types of AS changes highlighted. Anomalies are found by visually searching the dates for unusual patterns of lines. In this example session, a range of dates showing all the AS change types were examined until an anomaly was discovered on August 14th, 2000 (top row of the right image in Figure 5.4). The displayed AS change types were then changed until the type of anomaly was isolated (third column in the top right image in the figure). The order of this exploration was not as straight-forward as displayed in the sheet, as the session results in the bottom of the figure show. These results, which occurred during the AS exploration, show how each AS change was toggled on-and-off in sequence.

In order to support the model presented here, the original tool was modified in several stages. First, the types of visualization transforms used by the system and the



$s_0 = \langle p_0, r_0, t_0, \emptyset \rangle$ Initial Result (time-step 0)

$s_1 = \langle p_1, r_1, t_1, p_1(time) | \, p_0 \mapsto p_1 \rangle$ Time-step 300

$s_2 = \langle p_2, r_2, t_2, p_2(time) | \, p_1 \mapsto p_2 \rangle$ Time-step 500

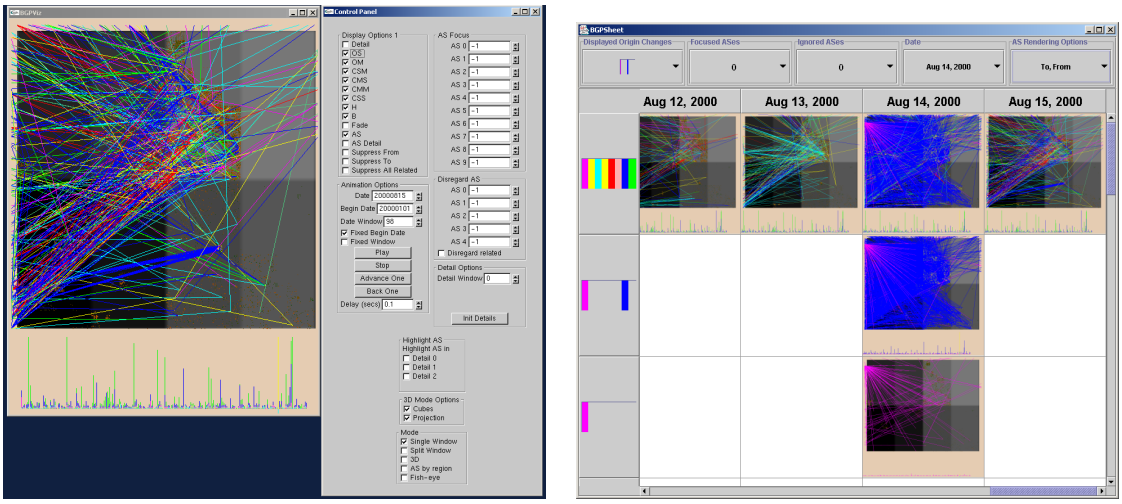
$s_3 = \langle p_3, r_3, t_3, p_3(resolution) | \, p_0 \mapsto p_3 \rangle$ Increase resolution

$s_4 = \langle p_4, r_4, t_3, p_4(resolution) | \, p_1 \mapsto p_4 \rangle$ Increase resolution

$s_5 = \langle p_5, r_5, t_3, p_5(resolution) | \, p_2 \mapsto p_5 \rangle$ Increase resolution

Figure 5.3: Representation of a multi-resolution, time-varying computational fluid dynamics visualization using the P-set Model. The user utilized the web-based spreadsheet-like interface introduced in Chapter 10. In this example, three results at different time-steps were rendered at the lowest resolution before the higher resolution images were rendered.

parameter and result types of each transform were determined. In this system, the major parameter types are the date, which of the eight AS changes to display, a list of ASes to highlight, a list of ASes to ignore, and options for modifying the display. Once the parameters were identified, the next stage determined how the parameter values can change. For this example, all parameter changes are discrete: there are no function applications and parameters cannot be manipulated over a range. Finally, hooks were added to the



... previous results ...

$s_{83} = \langle p_{83}, r_{83}, t_{83}, p_{83}(date) | \, p_{82} \mapsto p_{83} \rangle$ — First August 14th result

$s_{84} = \langle p_{84}, r_{84}, t_{84}, p_{84}(ASChange) | \, p_{83} \mapsto p_{84} \rangle$ — Toggle one change type off

$s_{85} = \langle p_{83}, r_{83}, t_{85}, p_{83}(ASChange) | \, p_{84} \mapsto p_{83} \rangle$ — Toggle the same change type on

$s_{86} = \langle p_{86}, r_{86}, t_{86}, p_{86}(ASChange) | \, p_{83} \mapsto p_{86} \rangle$ — Toggle another change type off

$s_{87} = \langle p_{83}, r_{83}, t_{87}, p_{83}(ASChange) | \, p_{86} \mapsto p_{83} \rangle$ — Toggle the same change type on

... etc. ...

Figure 5.4: An example of augmenting an existing visualization system (left) to store visualization session information using the model. The interface is a Border Gate Protocol (BGP) visualization tool; the right figure displays a spreadsheet view of an exploration process originally captured by the tool. Dates are shown across the columns and the types of origin AS change displayed (indicated by color) are shown down the rows. The session information (bottom) shows that the AS change type was determined by sequentially toggling the display of each change type.

user interface elements corresponding to each parameter type in order to capture changes in parameter values. These hooks update session information stored by a separate library. The library (written in Python) interfaces with the original tool (written in C++) in order to produce the representation used by the spreadsheet (written in Java, the top right image in Figure 5.4) to display the process. The above process can be repeated with other visualization systems in order to make use of the model.

5.4 Comparison

The transform model used by the P-set Model fixes the ambiguity in the data state model while introducing explicit parameter type modeling. In practice, this detail

in modeling is only needed in interfaces that can edit the transform (such as in data-flow visualization interfaces). The current representation of the model, discussed in Chapter 7, only uses the visualization transforms signature to identify the transform.

In comparison the previous session models, the P-set model has several advantages. This model addresses the visualization path model's inability to determine how results were derived from one another by introducing the parameter derivation calculus. Like the GDE Model, the P-set Model can have complex derivations (multiple ancestors or descendents), thus avoiding the single parent problem of the GRASPARC tree model. This model explicitly addresses how parameters were changed by the user, unlike the GDE model. The P-set fully captures the visualization exploration process.

5.5 Summary

The P-set Model for visualization exploration describes the method of visualization using an augmented data state model and encapsulates the visualization session using a new parameter derivation calculus. The formal foundation for the model allows different forms of analysis of visualization sessions. In the next chapter, visual analysis will be introduced via graphs of the visualization process. These graphs will be used to gain other types of insight into the visualization exploration process.

Chapter 6

Visualization Session Analysis

6.1 Introduction

During the visualization process, a user iteratively explores a very large space of visualization parameters in order to discover visualization results of interest. The search of this space can be costly—especially for very large data sets. By presenting the visualization process to the user, unnecessary re-explorations can be avoided. Such a presentation also assists the user understand and thus navigate the visualization space. Visualization process information is also useful for system designers: Representations of the process can highlight inefficiencies of the system or suggest common patterns of exploration. Methods for visualizing the visualization process are thus beneficial both in data exploration and system design.

This chapter describes new methods for extracting visual representations of the visualization process. Visualization sessions are themselves visualization using visualization process graphs introduced in this chapter. Different relations utilizing the P-set Model of visualization exploration are used to build the process graphs. The methods presented here allow effective visual analysis of different properties of the visualization exploration process.

Name	Relation	Description
History	*next-results*	Time-line of results
Session Result Derivation	*derives*	Temporally distinct derivations
P-set Derivation	*derives*	P-set derivations
P-set Difference	*differs-by-one*	Difference in p-set parameter values

Table 6.1: Visualization process relationships and the relations used.

6.2 Visualization Process Relationships

Before the visualization process can be presented visually, the relationships within the process must be extracted. The relationships are represented by (possibly directed) graphs—an edge exists between two session results or p-sets if they satisfy some relation. Different types of relationships emphasize different characteristics of the process. For example, one relationship between session results is their relative temporal ordering—i.e., which result was generated first. These relationships are measured using *visualization process relations*; these relations determine whether two nodes are connected (and the direction of the connection in a directed graph) based upon properties of the session results. In this work, four relationships and their corresponding relations are considered: History, Session Result Derivation, P-set Derivation, and P-set Difference (summarized in Table 6.1).

6.2.1 History Relationship

This relationship only uses the timestamps of the visualization session results. The purpose of this relation is to provide a temporal ordering of the results. It is important to know what results were generated after another and what results were generated during the same time-step. The *next-results* relation defined below satisfies this need.

Two relations are used—the *next-result* relation $\rightharpoonup$ and *same-timestamp* relation $\rightleftharpoons$. A third relation, *previous-result* $\leftharpoonup$ exists and is the inverse of *next-result*, but is not used in graph construction. The two relations are defined as follows (where $\mathcal{S} = (T, \mathsf{P}, R, S)$ is the visualization session under study):

Definition (*next-result* Relation). Given two session results $s_i = (p_i, r_i, t_i, \delta_i)$ and $s_j = (p_j, r_j, t_j, \delta_j)$, $s_i, s_j \in S$, $s_i \rightharpoonup s_j$ if and only if $t_j = t_i + 1$. In other words, s_j occurs

immediately after s_i temporally.

Definition (*same-timestamp* Relation). Given two session results $s_i = (p_i, r_i, t_i, \delta_i)$ and $s_j = (p_j, r_j, t_j, \delta_j)$, $s_i, s_j \in S$, $s_i \rightleftharpoons s_j$ if and only if $t_j = t_i$. In other words, s_j was generated during the same timestep at s_i.

The *next-result* relation is irreflexive and antisymmetric. The *same-timestamp* relation is reflexive and symmetric and thus transitive. Given these two relations, the *next-results* relation can be defined upon sets of session results:

Definition (*next-results* Relation). Given a visualization session $\mathcal{S} = (T, \mathsf{P}, R, S)$ and two sets of session results $S_{before} = (s_0, \ldots, s_n)$ and $S_{after} = (s'_0, \ldots, s'_m)$, $S_{before}, S_{after} \subseteq S$, $S_{before} \rightharpoonup S_{after}$ if and only if

1. $\forall s_i, s_j \in S_{before}$, $s_i \rightleftharpoons s_j$.
2. $\forall s \in S - S_{before}$, $\nexists s_i \in S_{before}$ such that $s \rightleftharpoons s_i$.
3. $\forall s'_i, s'_j \in S_{after}$, $s'_i \rightleftharpoons s'_j$.
4. $\forall s \in S - S_{after}$, $\nexists s'_i \in S_{after}$ such that $s \rightleftharpoons s'_i$.
5. $\forall s_i \in S_{before}$ and $\forall s'_j \in S_{after}$, $s_i \rightharpoonup s'_j$.

In other words, S_{before} and S_{after} each contain results generated during the same time-step and the results in S_{after} were generated in one time-step after those in S_{before}.

Like *next-result*, *next-results* is also irreflexive and antisymmetric. Note that the *next-results* relation is defined such that only sets of session results that contain all the results generated during the same time-step satisfy the relation. This property avoids potential ambiguity in the graphs based upon this relation.

6.2.2 Session Result Derivation Relationship

This relationship establishes parent-child relationships based upon result derivations. Though it is possible to generate the same visualization result more than once, each is considered a separate session result and thus will have distinct parent-child relationships.

If the relationships between p-sets is desired, the next relationship (p-set derivation) is relevant.

The *derives* relation $\Rightarrow$ forms the basis the session result derivation relationship. Formally, the *derives* relationship is defined as follows:

Definition (*derives* Relation (Session results version)). Given a visualization session $\mathcal{S} = (T, \mathsf{P}, R, S)$ and two session results $s_i = (p_i, r_i, t_i, \delta_i)$ and $s_j = (p_j, r_j, t_j, \delta_j)$ $(s_i, s_j \in S)$ such that δ_i, δ_j are parameter calculus instances of the form $d_k = (X_k, \mathsf{P}_{k,input}, \mathsf{P}_{k,output})$, $s_i \Rightarrow s_j$ if and only if:

1. $t_i < t_j$
2. At least one of the following holds:
   - $p_i \in \mathsf{P}_{j,input}$ (i.e., p_i is in d_j's input tuple)
   - $\exists p_i(k) \in p_i$ such that $\exists (l_m, P_{m,input}, P_{m,output}) \in X_j$ such that $p_i(k) \in P_{input,m}$ (i.e., one of p_i's parameter values must occur as an input parameter value in d_j).
   - $\exists p_i(k) \in p_i$ such that $\exists (l_m, P_{m,input}, P_{m,output}) \in X_j$ such that $p_i(k) \in P_{output,m}$ (i.e., one of p_i's parameter values must occur as an output parameter value in d_j).
3. $\forall k$ such that $i < k < j$, $\nexists s_k \in S$ such that s_k satisfies criteria 1 and 2 with respect to s_j. In other words, only the most recent session result with the same p-set is eligible to be s_j's ancestor.

A session result s_i derives result s_j if a parameter from s_i's p-set p_i was used by s_j's parameter calculus instance d_j in one of its parameter transform list elements or p_i is a member of d_j's input tuple. In addition, s_i must be the last session result with the p-set p_i recorded before s_j. Due to this restriction, session results with the same p-set but different timestamps are not related. In addition, session results generated with the same derivation, such as those from a function application, are not considered derived from each other. This relation is irreflexive and antisymmetric.

6.2.3 P-set Derivation Relationship

The p-set derivation relationship is similar to the session result derivation relationship, but it captures derivations between p-sets, not session results. If a user performed redundant exploration (i.e., generated the same result more than once), this will be reflected in the difference in cardinality of the session result derivation relation and the p-set derivation relation. Formally, the p-set *derives* relations is as follows:

Definition (*derives* Relation (P-set version)). Given a visualization session $\mathcal{S} = (T, \mathsf{P}, R, S)$ and two p-sets p_i, p_j $(p_i, p_j \in \mathsf{P})$, $p_i \Rightarrow p_j$ if and only if there exist session results $s_a = (p_i, r_a, t_a, d_a), s_b = (p_j, r_b, t_b, r_b) \in S$ such that $s_a \Rightarrow s_b$.

Like the session result version, p-set *derives* is irreflexive and antisymmetric.

6.2.4 P-set Difference Relationship

The final relationship only considers differences in session result p-sets. This measure is useful in determining the depth of the exploration—the more p-sets differ, the more the visualization parameter space has been explored. For this measure, the *differs-by-one* relation Δ_1 is used:

Definition (*differs-by-one* Relation). Given a visualization session $\mathcal{S} = (T, \mathsf{P}, R, S)$ and two p-sets $p_i = \{p_i(0), \ldots, p_i(n)\}, p_j = \{p_j(0), \ldots, p_j(n)\}$ $(p_i, p_j \in \mathsf{P})$, $p_i \Delta_1 p_j$ if and only if there exists $k, 0 \leq k \leq n$, such that $p_i(k) \neq p_j(k)$ and there does not exist $m, 0 \leq m \leq n, m \neq k$, such that $p_i(m) \neq p_j(m)$. In other words, p_i and p_j differ in only one parameter value.

This relation is irreflexive and symmetric.

If desired, a more restrictive version of the difference relation can be used. For example, a *differs-by-one-in-dataset* relation $\Delta_{(1,dataset)}$ could be defined as follows:

Definition (*differs-by-one-in-dataset* Relation). Given a visualization session $\mathcal{S} = (T, \mathsf{P}, R, S)$ and two p-sets $p_i = \{p_i(dataset), p_i(0), \ldots, p_i(n)\}, p_j = \{p_j(dataset), p_j(0), \ldots, p_j(n)\}$ $(p_i, p_j \in \mathsf{P})$, $p_i \Delta_{(1,dataset)} p_j$ if and only if $p_i(dataset) \neq p_j(dataset)$ and there

does not exist $k, 0 \leq k \leq n$, such that $p_i(k) \neq p_j(k)$. In other words, p_i and p_j differ in only the dataset parameter value.

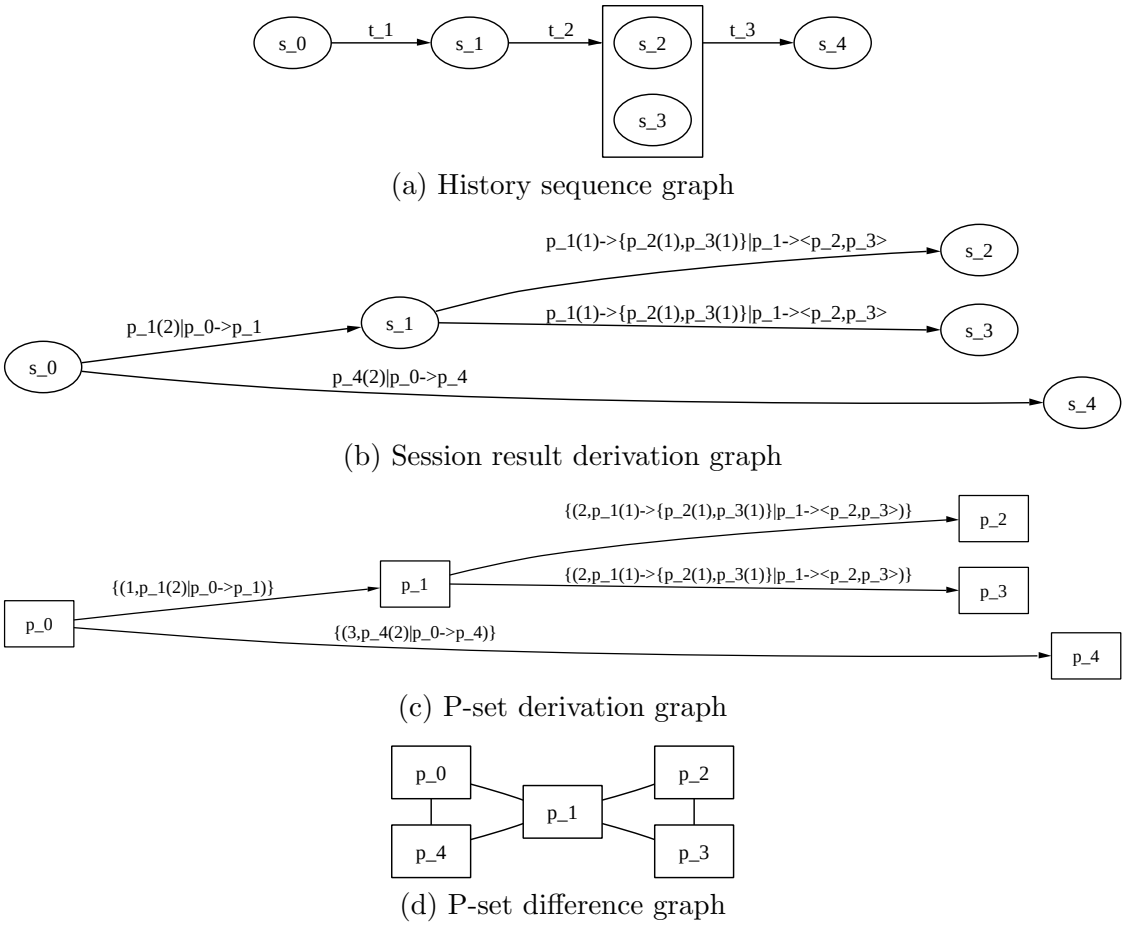
Similar relations can be defined for other parameter types.

6.2.5 Using Visualization Process Relationships

Each relationship highlights different aspects of the visualization process. Though parameter derivation information is not present in the history relations, they give a clear sense of the flow of time during the visualization process. Both the session result and p-set derivation relations are needed to get a sense of result parent-child relationships. The session result derivation relationship combines the sense of time the history relationship provides with a notion of p-set relationship. The p-set derivation relation can collapse some of the process structure, indicating where users returned to previous result (through cycles). Thus, the p-set relation finds where re-visiting occurred while the session result metric displays the temporal order of the re-visiting. Finally, the p-set difference metric gives a sense for the depth of exploration during the process. Shallow spanning trees of graphs using this metric signify a visualization process that did not deeply search the space of parameter values while deep spanning trees could suggest lack of focus. In fact, using these relations to build graphs is a powerful method for performing visualization session analysis. This idea will be explored more in-depth in the next section.

6.3 Visualization Process Graphs

Visualization process graphs visually summarize the visualization process; these graphs are similar in purpose and properties to the graphs of the GDE Model in Lee's thesis [37], though the p-set difference graph has no analog in Lee's work. The process graphs utilize the visualization session relations discussed previously in their construction. As such, there are four graphs of interest: The history sequence graph, the session result derivation graph, the p-set derivation graph, and the p-set difference graph. Figure 6.1 depicts these graphs for the example session from Figure 5.1.



(a) History sequence graph

(b) Session result derivation graph

(c) P-set derivation graph

(d) P-set difference graph

Figure 6.1: Visualization process graphs for the visualization sessions results (the s_i) and p-sets (the p_i) from Figure 5.1. The graphs provide an "at-a-glance" overview of the visualization process. The graphs clearly show that s_4 is not descended from s_2 or s_3, whereas that information may not be immediately apparent from the session results.

6.3.1 History Sequence Graph

In the history sequence, branches in the visualization process are collapsed into a single element. Each element in the sequence is a set of session results that were generated by the user in a single operation. These elements are drawn from the domain and range of the *next-results* relation. The sequence can be displayed graphically using vertices representing the sets of session results created during a single time step and directed edges representing the flow of time. These edges are labeled with the timestamp to order the sequence. Formally,

Definition (History Sequence Graph). Given a visualization session $\mathcal{S} = (T, \mathsf{P}, R, S)$, a history sequence graph $G_{\rightharpoonup} = (V_{\rightharpoonup}, E_{\rightharpoonup})$ is a labeled graph with labels $L_{\rightharpoonup} : E_{\rightharpoonup} \mapsto l$ such that:

- $V_{\rightharpoonup} = \{S' \subseteq S : \exists S'', S'' \subseteq S$ such that $(S', S'') \in$ *next-results* $\vee\, (S'', S') \in$ *next-result*$\}$.
- $E_{\rightharpoonup} =$ *next-results*
- $\forall e = (S', S'') \in E_{\rightharpoonup}, L_{\rightharpoonup}(e) = t$.

For most visualization interfaces, the history sequence graph is a line. This is due to the fact that most of these interfaces cannot generate more than one result at a time. Towards this end, a more informative graph that displays both derivation and temporal information is needed. Such a graph is the session result derivation graph.

6.3.2 Session Result Derivation Graph

The history sequence is insufficient for describing the relationships between session results. The sequence does not distinguish between results derived directly from their predecessor or those derived from earlier results. These relationships are vital to understanding the entirety of the visualization process and are captured by the session result derivation graph. This directed graph is defined as follows:

Definition (Session Result Derivation Graph). Given a visualization session, $\mathcal{S} = (T, \mathsf{P}, R, S)$, a session result derivation graph $G_{S \Rightarrow S} = (V_{S \Rightarrow S}, E_{S \Rightarrow S})$ is a labeled, directed graph with labels $L_{S \Rightarrow S} : E_{S \Rightarrow S} \mapsto l$ such that:

- $V_{S \Rightarrow S} = S$. In other words, the vertices are session results in $\mathcal{S}$.
- $E_{S \Rightarrow S} =$ *derives* (session version). In other words, an edge exists between session results s_i and s_j $(s_i, s_j \in S)$ if and only if $s_i \Rightarrow s_j$.
- $\forall e = (s_i, s_j) \in E_{S \Rightarrow S}, L_{S \Rightarrow S}(e) = \delta_j$. The edges are labeled with the corresponding derivation using the parameter derivation calculus.

Derivations that generated or used several results are clearly identified in the graph. It is possible for the graph to contain disconnected components. Each disconnected

component corresponds to a different visualization transform as there can be no derivations between transforms. These properties are shared by both the session result derivation graph and the p-set derivation graph (next section).

An important property of the session result derivation graph is that it is a collection of directed, acyclic graphs (DAGs). There are no cycles because of the ordering enforced by the derivation relation—no result can derive a result with a lower or same timestamp value. The DAGs represent the derivations related to a single visualization transform. Each DAG possesses a node corresponding to the default result of the visualization transform. The default result is the session result that corresponds to the initial parameter values for a visualization transform; if there is no appropriate default value for a certain parameter (such as a data set), then an "undefined" value is used. By convention, any completely new set of parameter values is derived from this default set. Only the default results are not derived from any other result.

6.3.3 P-set Derivation Graph

The p-set derivation graph shares many properties with the session result derivation graph. Both are labeled, directed graphs with possible unconnected components. However, unlike the session result graphs, p-set derivation graphs can have cycles. This properties is a consequence of the lack of temporal ordering in the p-set derivation. These cycles serve an important purpose—they indicate points of redundant exploration. If a p-set is revisited, then its corresponding visualization result was generated multiple times. This could be a potentially costly operation, and thus cycles should be avoided. By using the p-set derivation graph, cycles can be detected, leading to a determination of their cause.

The p-set derivation graph's formal definition is similar to the session results graphs' definition, differing mainly in the vertex type (p-sets instead of session results) and the labels:

Definition (P-set Derivation Graph). Given a visualization session, $\mathcal{S} = (T, \mathsf{P}, R, S)$, a p-set derivation graph $G_{\mathsf{P}\Rightarrow\mathsf{P}} = (V_{\mathsf{P}\Rightarrow\mathsf{P}}, E_{\mathsf{P}\Rightarrow\mathsf{P}})$ is a labeled, directed graph with labels $L_{\mathsf{P}\Rightarrow\mathsf{P}} : E_{\mathsf{P}\Rightarrow\mathsf{P}} \mapsto l$ such that:

- $V_{\mathsf{P}\Rightarrow\mathsf{P}} = \mathsf{P}$. In other words, the vertices are p-sets in $\mathcal{S}$.
- $E_{\mathsf{P}\Rightarrow\mathsf{P}} =$ *derives* (p-set version). In other words, an edge exists between p-sets p_i and p_j in P if and only if $p_i \Rightarrow p_j$.
- $\forall e = (p_i, p_j) \in E_{\mathsf{P}\Rightarrow\mathsf{P}}, L_{\mathsf{P}\Rightarrow\mathsf{P}}(e) = \{(t_m, \delta_m) : \exists n < m \wedge s_n \Rightarrow s_m \wedge p_i \in s_n \wedge p_j \in s_m\}$. The edges are labeled with the corresponding timestamps and derivations for each derivation from p_i to p_j.

Unlike the session result derivation graph, the p-set derivation graph must include derivation information from several different time steps per edge due to the possibility of cycles. Alternatively, a unique edge for each derivation could be used (this would require that $E_{\mathsf{P}\Rightarrow\mathsf{P}}$ be a multiset instead of a set). If there are no cycles, then the session result derivation graph and p-set derivation graph are isomorphic.

6.3.4 P-set Difference Graph

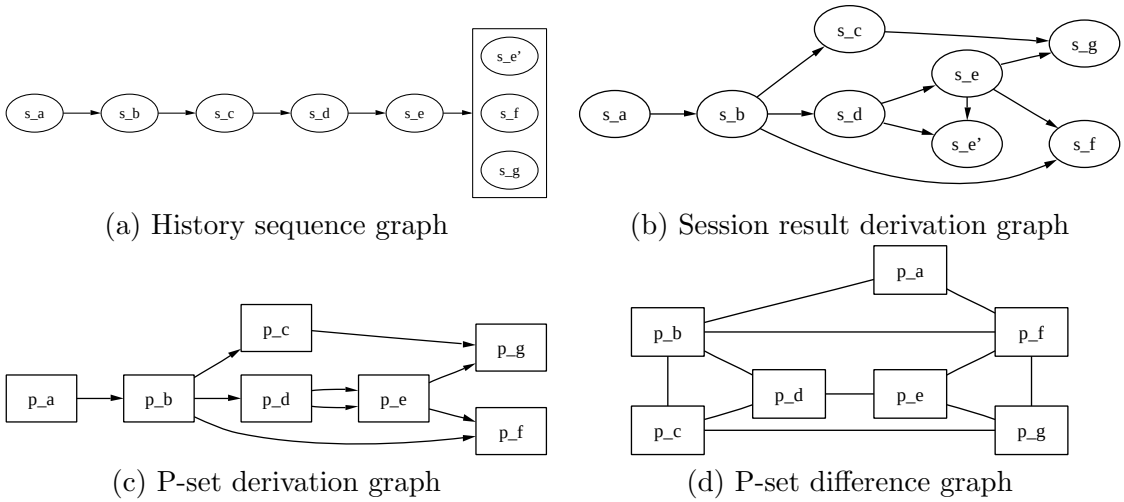
The p-set difference graph highlights the depth of exploration whereas the other graphs highlight the structure of the exploration. Each cluster in a p-set difference graph represents similar results—results where the p-set only differed by one value. The more cluster there are, the larger the explored parameter space.

Like the p-set derivation graph, the p-set difference graph uses p-set as vertices. Unlike the other graphs, it is an undirected graph since its underlying relation is reflexive:

Definition (P-set Difference Graph). Given a visualization session, a p-set difference graph $G_{\Delta_1} = (V_{\Delta_1}, E_{\Delta_1})$ is an undirected graph such that:

- $V_{\Delta_1} = P$.
- $E_{\Delta_1} =$ *differs-by-one*.

As defined, the p-set difference graph is unlabeled. If desired, the edges could be labeled with the parameter type that differs between the two results.



(a) History sequence graph

(b) Session result derivation graph

(c) P-set derivation graph

(d) P-set difference graph

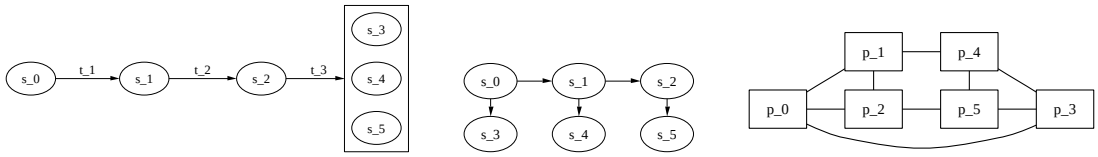
Figure 6.2: Visualization process graphs sans labels for the Image Graph visualization session in Figure 5.2.

6.4 Examples

To further illustrate the visualization process graphs, the examples from Section 5.3 are revisited, this time displaying their corresponding process graphs.

6.4.1 Image Graph Example

Figure 6.2 depicts the visualization process graphs for the Image Graph vessel visualization session. The figure illustrates the various nuances of the process metrics. Like the previous example (Figure 6.1), the history session graph displays a cluster of results. Again, this signifies the generation of several results in the same time step (in this case, the last three results). More interesting in this example is the difference between the session result derivation graph (Figure 6.2b) and the p-set derivation graph (Figure 6.2c). In the session, p-set p_e was generated twice: First when created during time step t_4 (s_e); the second during the parameter propagation during time step t_5 ($s_{e'}$). Though the two graphs are different, a cycle is not present. However, it is a redundant exploration. Finally, in the p-set difference graph, there are two major clusters—one on the left (p_b–p_d) and one on the right (p_e–p_g). These clusters are mirrors of each other. This is due to the fact that p-sets p_b and p_f differ only in their zoom value from p_a while every other results differs in both



(a) History sequence (b) Session derivation (c) P-set difference

Figure 6.3: Visualization process graphs sans labels for the web-based VisSheet visualization session in Figure 5.3.

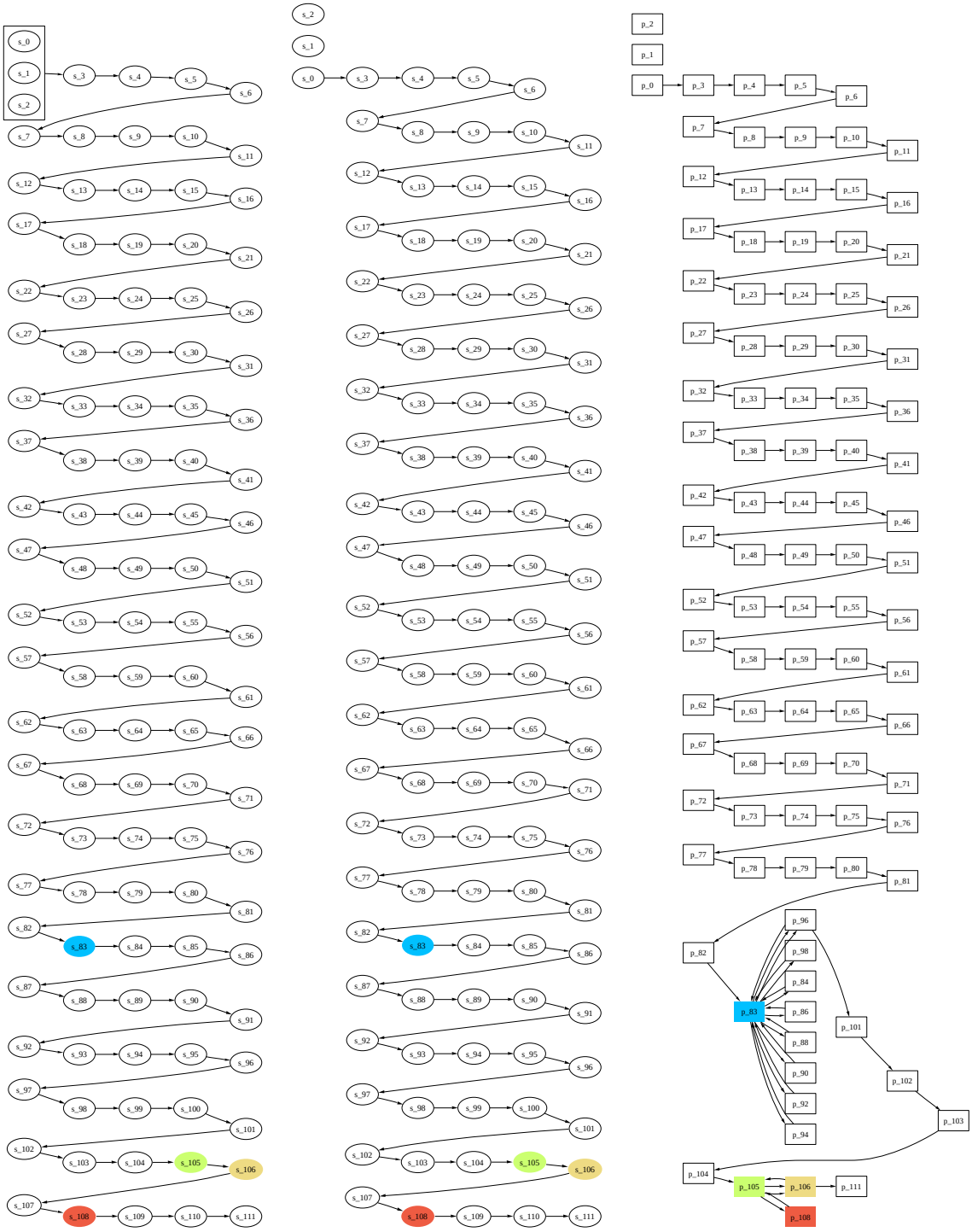
zoom and view position. The results on the left-hand side share the smaller zoom factor, the results on the right-hand side share the higher zoom factor. It is also interesting to note that the horizontal levels (save the first with p_a) share a view position distinct from those on other levels. Thus, the p-set difference graph also provides some information on structure in the exploration.

6.4.2 VisSheet Example

The VisSheet structures visualization exploration such that there no result is created more than once. Characteristic of the VisSheet are the process graphs from Figure 6.3. Since there was no redundant exploration, the session result derivation and p-set derivation graphs are the same. The session result derivation graph (Figure 6.3b), with is grid-like structure, is indicative of VisSheet-based exploration. Breaks in this structure only occur when a displayed parameter is changed (creating a "bridge" link between two grids) or when an operator is applied. Similarly, clusters for each row and column in the sheet also exists in the p-set difference graph, each connected to their row and column neighbors.

6.4.3 Dynamic Manipulation Interface Example

This last example analyzes the complex session from the an Internet routing anomaly visualization tool. Recall that the tool displays different types of changes to ownership of ASes over time using colored lines or cubes depending on the visualization used. Typically, the anomalies are found by stepping quickly through the dates, using visual pattern matching to find unusual changes. The visualization process graphs in Figures 6.4 and 6.5 corroborate this exploration pattern.



(a) History sequence (b) Session derivation (c) P-set derivation

Figure 6.4: Visualization process graphs sans labels for the Internet routing visualization session in Figure 5.4. Colors correspond to important results displayed in Figure 6.6.

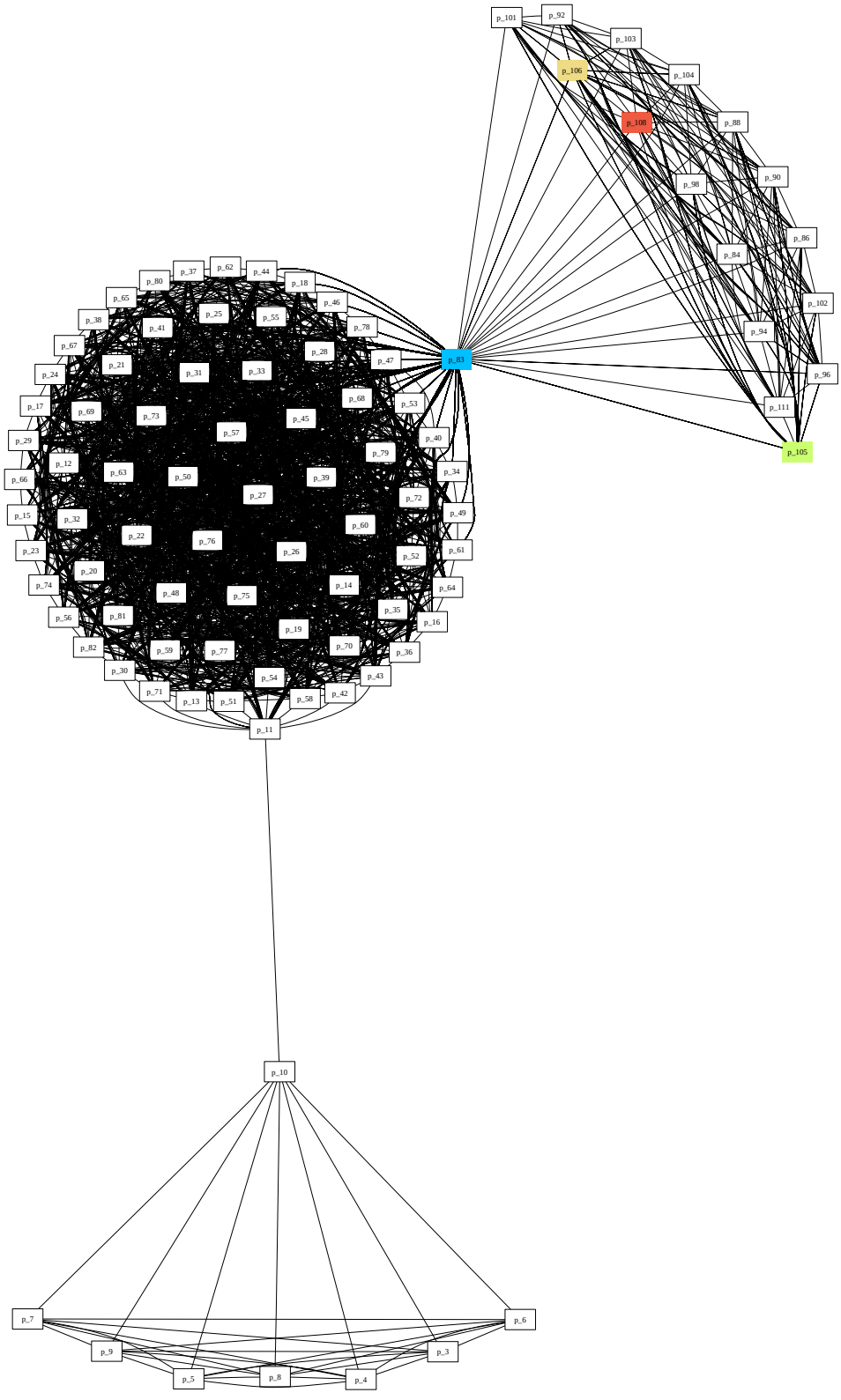
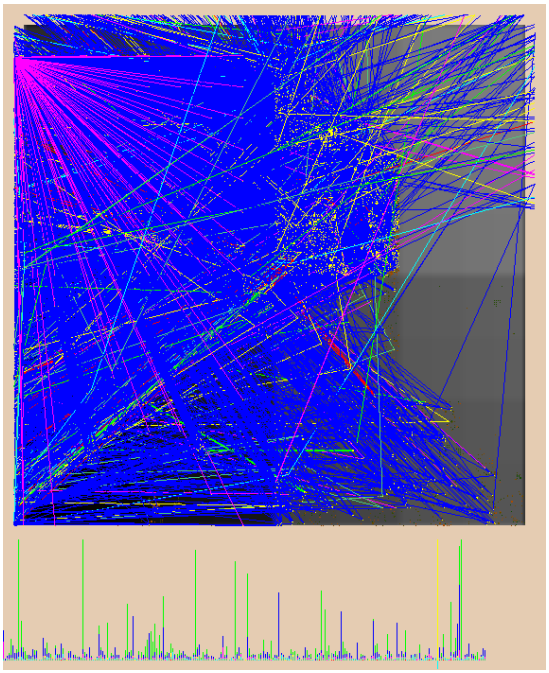
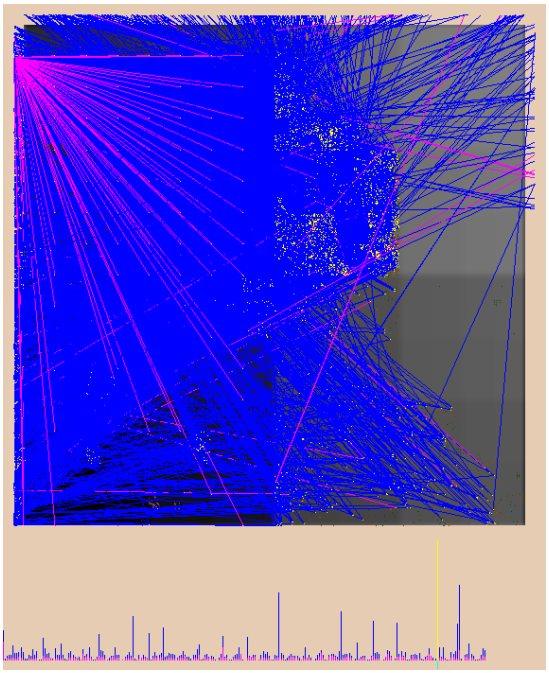


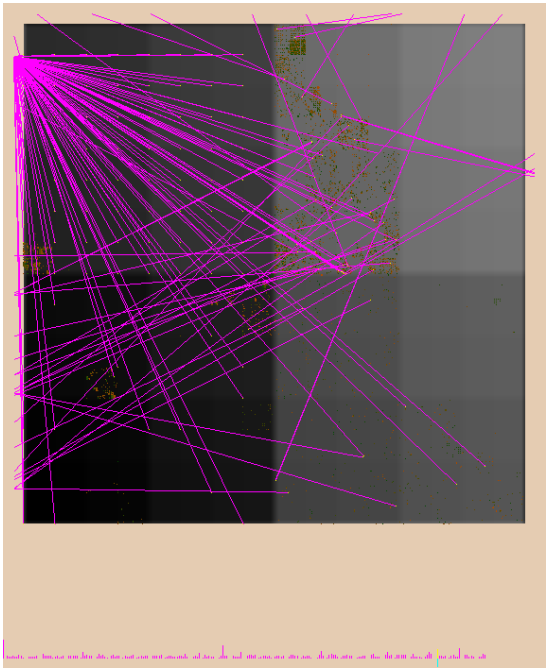
Figure 6.5: P-set difference graph for the Internet routing visualization session in Figure 5.4. Colors correspond to important results displayed in Figure 6.6.



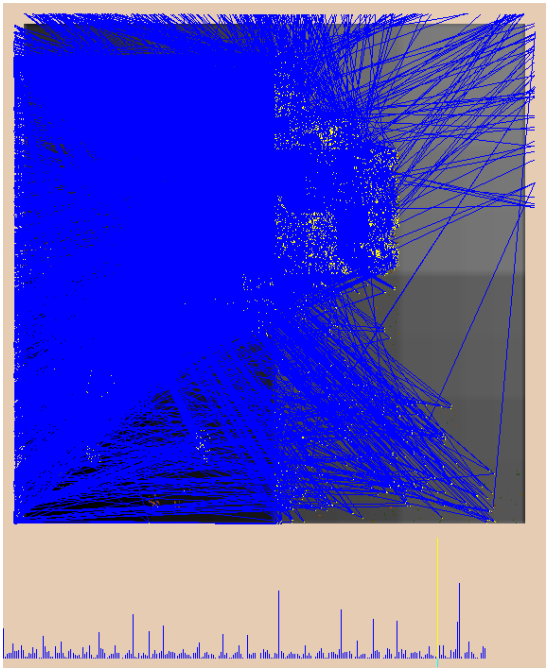
(a) r_{83}: Anomalies detected (blue)



(b) r_{105}: Major anomalies isolated (green)



(c) r_{106}: First anomaly (yellow)



(d) r_{108}: Second anomaly (red)

Figure 6.6: Important results from the BGP exploration session and their corresponding colors in Figures 6.4–6.5.

The history sequence (Figure 6.4a) illustrates that multiple results were only created during the first time step. These results correspond to the default results for the

three visualization transforms (2D, 3D, focus+context 2D) that the tool exposes; the disconnected nodes in the other graphs belong to the later two transforms. The tool does not otherwise allow multiple results to be generated. The difference between the session result and p-set result graphs (Figure 6.4b–c) indicate that redundant exploration occurred. The major divergence occurs at session result s_{83}—the first result that explored the date of the AS ownership error (Figure 6.6a and the blue node in Figures 6.4–6.5). To determine the type of anomaly, the user then toggled the different displayed AS change types off-and-on individually (the cluster connected to p_{83} in Figure 6.4) until the two anomalies were isolated (p-set p_{105} with corresponding result Figure 6.6b [green in Figures 6.4–6.5]). These were then displayed in isolation (p-sets p_{106} and p_{108}, displayed in Figure 6.6c–d [yellow and red respectively in Figures 6.4–6.5]). The exploration then ended.

The p-set difference graph (Figure 6.5) provides a different view of the process. In the graph, the three clusters correspond to the three phases of the exploration: setup, exploration, and investigation. In the first phase (the bottom cluster), the user set the initial parameters in order to display all the different AS change types for visual comparison. This phase ended with p-set p_{10}, after which the second phase begun. In this phase (p-sets p_{11}–p_{83}, middle of the figure), the viewed date was changed repeatedly. By the size of the cluster, most of the time was spent in this section. The final cluster (p-sets p_{83}–onward, top-right) indicates the final phase where the AS change type displayed was toggled until the anomalies were isolated. Combined with the other process graphs, visual analysis of visualization sessions gives insight into the user's experience.

6.5 Process Graph Analysis

As demonstrated, visualization process graphs allow in-depth analysis of the visualization process. To assist in this analysis, visualization process metrics and common patterns can be used.

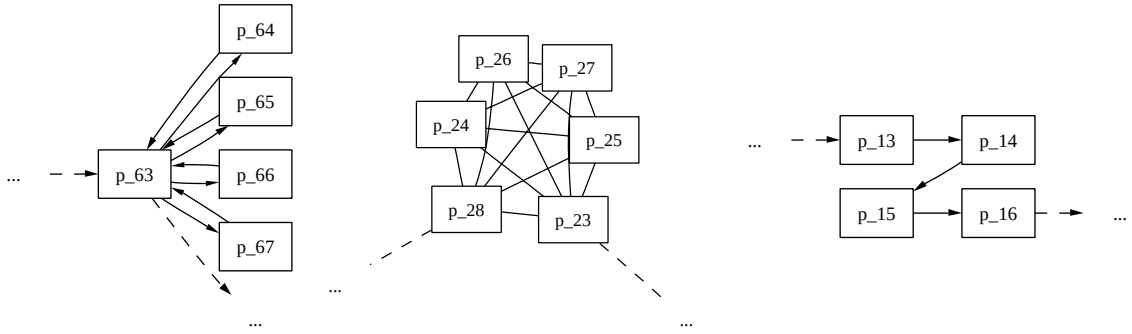
6.5.1 Metrics

Visualization process metrics measure aspects of the visualization session. Metrics can act upon the formal description of the session directly or upon views of that session such as the process graphs. For example, Lee's thesis [37] discusses process graph metrics for the GDE model that can be adapted to several of the process graphs discussed in this work. As another example, consider two metrics to measure the breadth and depth of exploration:

- *P-Set Breadth Metric* Given a visualization session $\mathcal{S}$, the value of breadth metric is the maximum number of neighbors of any node in G_{Δ_1}, the p-set difference graph.
- *P-set Depth Metric* Given a visualization session $\mathcal{S}$, the value of the depth metric is the number of nodes in the longest shortest path between any two nodes in G_{Δ_1}, the p-set difference graph.

By these metrics, the exploration depicted in Figure 6.5 is rather broad (due to the large central cluster) but not very deep (for the same reason). Metric analysis of sessions is fruitful area of future research.

6.5.2 Patterns



(a) Hunt-and-Peck (b) Pattern Burst (c) Flat-Line Derivation

Figure 6.7: Different visual patterns exhibited by visualization process graphs.

Another approach in session analysis is to search for visual patterns in the process graphs. Such patterns can give provide information about the interface used or the intention of the user. Consider the following patterns illustrated in Figure 6.7:

- *Hunt-and-Peck* In this pattern, a user repeatedly changes a single parameter type, looking for a particular feature. This pattern is indicated by a cluster of cycles in the p-set derivation graph (Figure 6.7a). This behavior is found in the BGP exploration in Figure 6.4c.

- *Parameter Burst* A parameter burst appears as a fully-connected cluster of sequentially numbered p-sets in the p-set difference graph (Figure 6.7b). This type of pattern indicates an exploration pattern that modifies a single parameter type until its desired value is found before moving on to the next parameter. Chains of bursts correspond to sequential explorations of different parameters types (as in Figure 6.5). This type of pattern often occurs in interfaces that cannot generate multiple results at a time.

- *Flat-Line Derivation* A flat-line derivation occurs in either the session result derivation graph or the p-set derivation graph when there are no branches in the graph (Figure 6.7c). If it occurs in the former but not the latter, then this indicates cycles and thus redundant exploration. If it occurs in the p-set derivation graph as well, it indicates that only the immediate previous result was ever used in a derivation. This pattern often indicates an unsophisticated user interface that does not allow access to previous results (in order to branch the exploration).

These patterns are useful to providing heuristics about the process without performing a full metric analysis. Also, by identifying these patterns, common behavior and problems with visualization explorations or visualization interfaces can be identified. Like the process metrics, further research into these patterns is called for.

6.6 Summary

The information stored within the visualization process is fairly complex. To gain an understanding of visualization sessions—and perhaps a better understanding of the data originally visualized—this information needs to be visualized. Four different graphs examining different parts of the visualization process were presented, and an in-depth example demonstrated the uses of this kind of visualization.

There are several potential applications for the work presented here. Visualizations of the visualization process give insight into the process that can be used in different ways. For example, the analysis of the BGP visualization session suggests that a more intuitive mapping of AS change type to color, a more effective parameter manipulation control, or better user training could be needed to remove the re-rendering cycle discovered. In addition, collaborators can use such visual representations of the visualization process to familiarize themselves with previous explorations. This exploration could in turn suggest further avenues of pursuit with the data under study.

In order for collaborators to utilize the information stored in the P-set Model and perform the analysis outlined here, a common representation of the model is needed. The next chapter discusses this representation.

Chapter 7

Representation

In order to share visualizations captured by the P-set model, a common data format is required. To be effective, the format must be extensible to different visualization applications. It is also desirable that the representation can be used by data-mining or analysis tools. These goals are accomplished by using XML to express the visualization process. This chapter presents an XML representation of the visualization process.

7.1 The P-set Model Representation

The Extended Mark-Up Language (XML [4]) is a standard data exchange format. Standardized technologies exist to parse and extract the content from XML documents. XML documents can also be transformed into HTML [16]. By expressing the visualization session with XML, the session can be easily shared with collaborators. Specific systems can translate their internal representation into the P-set Model (via XML) which can then be translated again into a representation usable by some other system.

The XML representation of the model is partitioned into five sections. The first section describes the visualization transforms used by listing their signatures (the parameter and result types) and name. In the next section, a list of the parameter values used is stored, each uniquely identified. Each distinct p-set is then recorded by identifying the parameter values composing the p-set. The p-sets are also given a unique identifier. Next, the visualization transform results are stored. For each result, a reference to the p-set

which generated it, an identifier, and the result itself are recorded. Note, for interactive systems which can generate results continuously over a range, only the first and last result in that range are stored. It is assumed that the interim results can be generated by interpolating over the parameter that varied. If two results are not sufficient, then "key frame" results—results where interpolation over the range is sufficient—could also be stored. In the final section, the visualization session's results are themselves stored. Each session result identifies its p-set, visualization result, timestamp, and the derivation information for that session result. Figure 7.1–7.2 provides a document type description (DTD [4]) that describes the representation while Figure 7.3 provides a visual depiction of the schema for the representation's DTD.

When generating the XML session document, there are different approaches to how parameters and results are stored. It is possible to embed a representation of these items directly into the XML representation. For large or binary elements, such as the data set used or the results themselves, this approach may be inadvisable. Instead, each parameter or result element in the XML document can provide an optional link attribute. A link is a URL describing where the actual parameter or result may be obtained. Linking can be used to reference large data sets over the network while accessing image files locally, avoiding costly transfers.

Note that the main purpose of the XML description of the model is for transport, not analysis. Analysis is performed on the information encoded by XML (the visualization session results from this model), not on the XML document itself. Given an XML document representing a visualization session, tools are expected to parse the document into their own internal structures before operating on the visualization session information.

7.2 Using the Representation

The representation serves two purposes. It allows sessions to be communicated between different visualization systems and can be used to analyze the process. For example, in the example in Figure 5.4, the representation was used to transport the visualization session from the Internet routing visualization tool to the VisSheet; a summarized version

```
<!ELEMENT visualization (transforms,parameter_values,result_values,session)>

<!ELEMENT transforms (transform)+>

<!ELEMENT transform (parameter_type+,result_type)>

<!ELEMENT parameter_type EMPTY>
<!ATTLIST parameter_type
  id ID #REQUIRED
  name CDATA #REQUIRED>

<!ELEMENT result_type EMPTY>
<!ATTLIST result_type
  id ID #REQUIRED
  name CDATA #REQUIRED>

<!ELEMENT parameter_values (parameter_value)*>

<!ELEMENT parameter_value #PCDATA>
<!ATTLIST parameter_value
  id ID #REQUIRED
  type IDREF #REQUIRED
  link CDATA #IMPLIED>

<!ELEMENT parameter_sets (parameter_set)*>

<!ELEMENT parameter_set (parameter)+>
<!ATTLIST parameter_set
  id ID #REQUIRED>

<!ELEMENT parameter EMPTY>
<!ATTLIST parameter
  value IDREF #REQUIRED
  pset IDREF #IMPLIED>

<!ELEMENT result_values (result_value)*>

<!ELEMENT result_value #PCDATA>
<!ATTLIST result_value
  id ID #REQUIRED
  type IDREF #REQUIRED
  pset IDREF #REQUIRED
  link CDATA #IMPLIED>

<!ELEMENT session (session_result)*>

<!ELEMENT session_result (unrelated|derivation)>
<!ATTLIST session_result
  id ID #REQUIRED
  pset IDREF #REQUIRED
  result IDREF|undefined #REQUIRED
  timestamp CDATA #REQUIRED>
```

Figure 7.1: DTD for the P-set Model representation. Note that the `parameter_value` and `result_value` elements are allowed to have additional attributes for encoding purposes.

```
<!ELEMENT unrelated EMPTY>

<!ELEMENT derivation (calculi,input_psets,output_psets)>
<!ATTLIST derivation
  id ID #REQUIRED>

<!ELEMENT calculi (calculus)+>

<!ELEMENT calculus (input_parameters,output_parameters)>
<!ATTLIST calculus
  type (application|range|function) #REQUIRED>

<!ELEMENT input_parameters (parameter)*>
<!ELEMENT output_parameters (parameter)+>

<!ELEMENT input_psets (pset)+>
<!ELEMENT output_psets (pset)+>

<!ELEMENT pset EMPTY>
<!ATTLIST pset
  value IDREF #REQUEST>
```

Figure 7.2: DTD for the P-set Model representation (continued).

of the representation can be found in Figure 7.4. In addition, since the representation is XML, it can be easily translated into HTML, such as for the vessel visualization depicted in Figure 7.5. This sort of overview is important in order to understand the visualization process without having to load the session into a visualization system.

One concern is the growth of the XML representation as visualization sessions become longer. In the worst case, the size of the file can increase quadratically with the number of results (if every new result is derived from all previous results—an unlikely case). However, in practice, the XML document does not approach anywhere near the size of the original data set for common large data sets and can be effectively compressed if needed. In the BGP example, the XML session encoding never exceeded a megabyte in size for over eighty session results. Combined with the binary PNG images for the rendered results, the overall size was 6.1 MB.

To this point, the model and representation discussed have focused on capturing the details of the visualization process independent of the user interface used. However, without a properly designed interface, the model cannot be used to its full effectiveness. In the next part, principles for such interfaces are discussed and demonstrated.

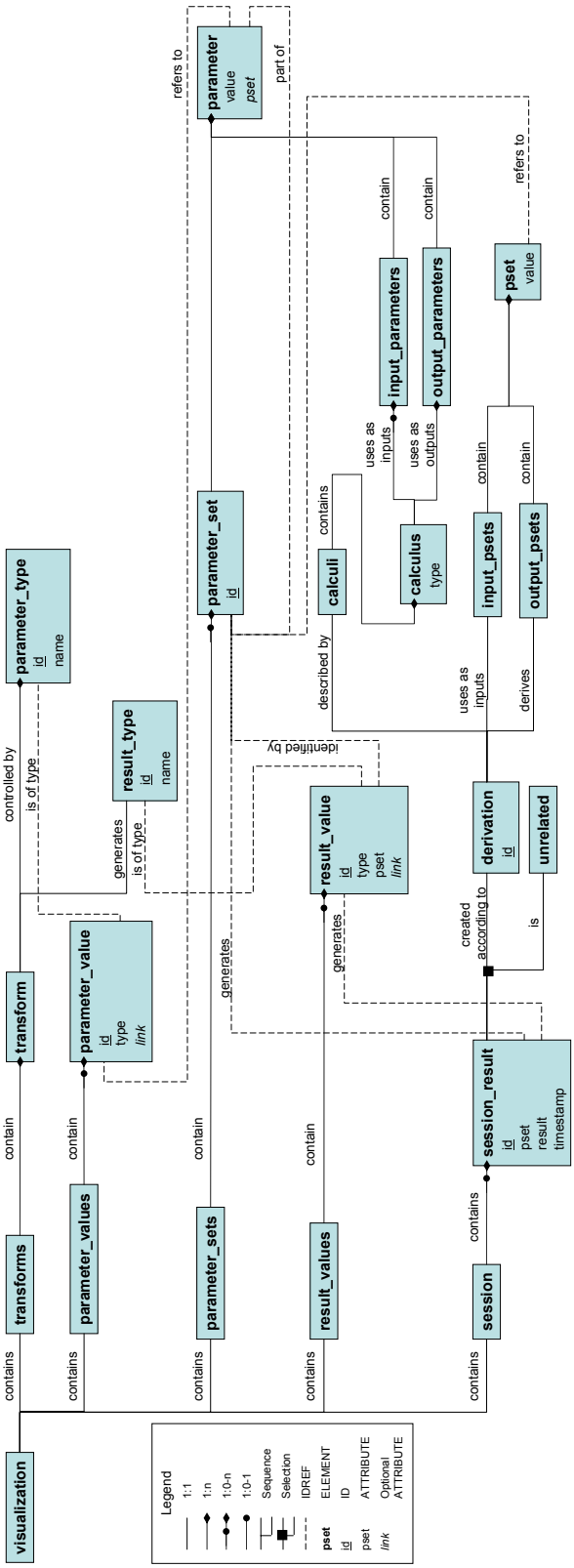


Figure 7.3: Visual representation of the schema for the P-set Model representation.

```
<?xml version="1.0" standalone="yes" ?>
<visualization>
  <transforms>
    <transform id="trans0">
      <parameter_type id="ptype0" name="Displayed Origin Changes" />
      <parameter_type id="ptype1" name="Focused ASes" />
      <parameter_type id="ptype2" name="Ignored ASes" />
      <parameter_type id="ptype3" name="Date" />
      <parameter_type id="ptype4" name="AS Rendering Options" />
      <result_type id="rtype0" name="8-bit RGBA Image" />
    </transform>
    ...
  </transforms>
  <parameter_values>
    <parameter_value id="param0" type="ptype7" >(0.5, 0.5)</parameter_value>
    <parameter_value id="param1" type="ptype0" >(0, 0, 0, 0, 0, 0, 0, 0)</parameter_value>
    ...
    <parameter_value id="param17" type="ptype4" >(1, 1, 1)</parameter_value>
    <parameter_value id="param18" type="ptype3" >20000101</parameter_value>
    ...
  </parameter_values>
  <parameter_sets>
    <parameter_set id="pset0" >
      <parameter value="param18" />
      <parameter value="param34" />
      <parameter value="param35" />
      <parameter value="param1" />
      <parameter value="param16" />
    </parameter_set>
    ...
    ⋮
    ...
  </parameter_sets>
  <result_values>
    <result_value id="result0" type="rtype0" pset="pset3" >'result0.png'</result_value>
    ...
  </result_values>
  <session>
    <session_result id="step0" pset="pset0" result="undefined" timestamp="0" >
      <unrelated />
    </session_result>
    <session_result id="step3" pset="pset3" result="result0" timestamp="1" >
      <derivation id="derivation0" >
        <calculi>
          <calculus type="application" >
            <input parameters>
            </input parameters>
            <output parameters>
              <parameter value="param19" />
              <parameter value="param2" />
            </output parameters>
          </calculus>
        </calculi>
        <input psets>
          <pset value="pset0" />
        </input psets>
        <output psets>
          <pset value="pset3" />
        </output psets>
      </derivation>
    </session_result>
    ...
    ⋮
    ...
  </session>
</visualization>
```

Figure 7.4: The XML representation of a visualization session using the BGP visualization tool; similar portions of the document are condensed for illustration purposes. The first section (blue) details what visualization transforms were used. The next portion (green) lists all the parameter values explored, followed by the parameter sets constructed (pink) and the visualization results rendered (red). Finally, the session information is recorded (purple). This representation encodes all the information described by the model and is used to transport session information between visualization systems.

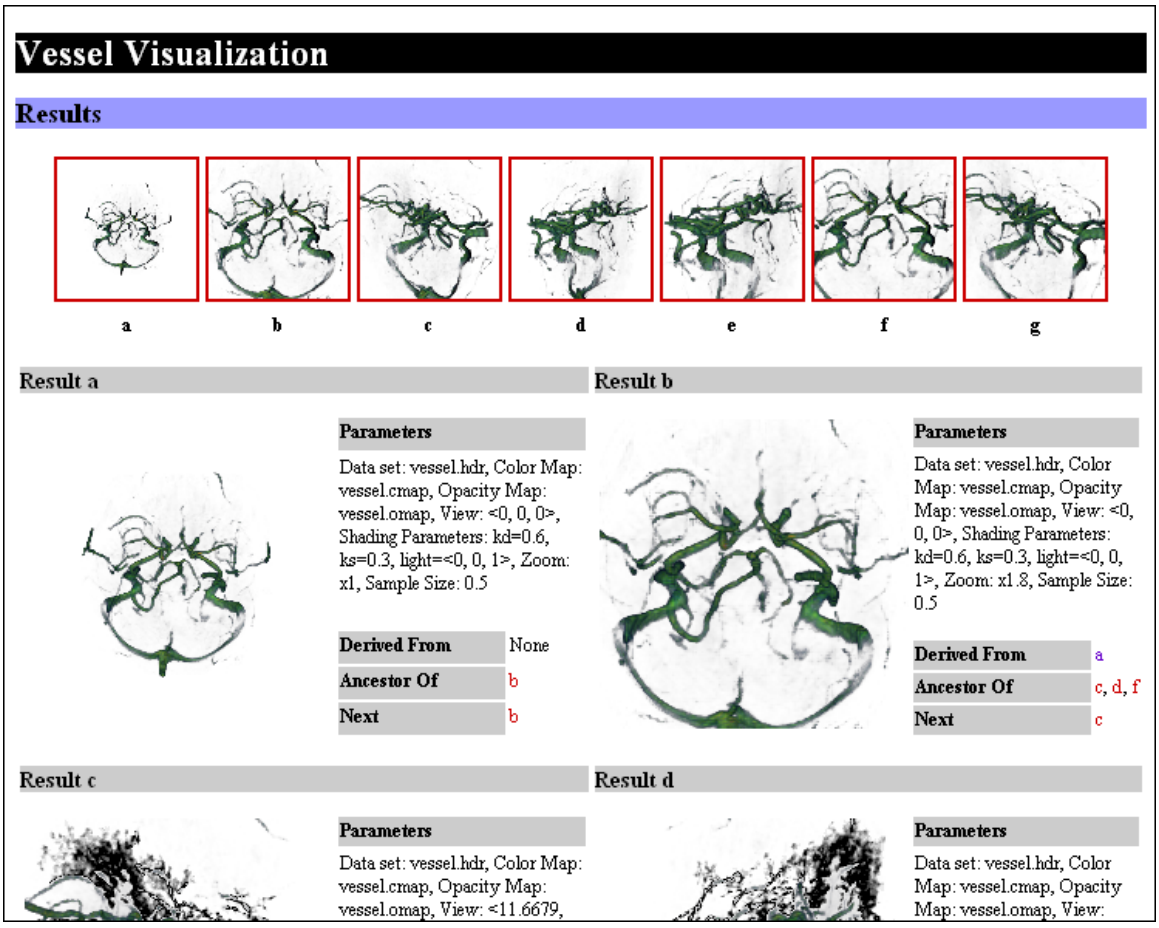


Figure 7.5: HTML overview of the vessel visualization session depicted in Figure 5.2.

Part III

External Representation

Chapter 8

Principles for Visualization Exploration Interfaces

Visualization user interfaces are used to assist the user in extracting insight from features of interest within their data. The interface hides the complexity of the techniques used during the visualization process from the user. Users control the visualization process through the user interface by applying operators on the data. Parameter settings determine the outcome of these operators. Thus, visualization user interfaces act in part as an intermediary between the data and the user.

Most current research in the field of visualization has focused on developing improved visualization techniques rather than improving the visualization user interfaces. Considering the massive data sets being generated by large research projects, this is understandable. Yet, at some stage, the user will need to interact with their data. Without a structured environment to apply new visualization techniques, these techniques cannot be properly utilized. Visualization user interfaces supply this structure.

8.1 Components of User Interface Design

User interface design incorporates a wide range of research topics. Elements include the graphical representation, input device specifications, user models, color and perceptual issues, ergonomics, etc. Domik and Gutkauf [17] define four component models

for visualization systems: the data model, the resource model, the user model, and the domain/task model. Data models perform the task of communicating the contents of possibly heterogeneous data formats to the user/system. The resource model describes the physical limitations of the hardware and software housing the visualization system; it also includes environmental factors which can affect the visualization process—e.g., the room lighting for workbench-based virtual reality environments. The user model handles the perceptual, memory, and coordination issues of the user. Finally, the domain/task model describes how properties of the scientific discipline affect the system's design. For example, most disciplines have conventions for the display of their data. This work focuses on the domain/task part of visualization system design common to all disciplines: the control of the visualization process, the display of this process, and the utilization of information from the discipline within the system. User modeling, system resource, and hardware issues will not be discussed.

Many of the more advanced visualization user interfaces can be classified as visual programming environments (VPEs) [6]. VPEs use visual expressions (spatial relationships, glyphs, text, animation) to control the underlying program execution. Data-flow and spreadsheet interfaces are common VPEs. The visual language community has developed several design principles for VPEs that can be applied to visualization user interfaces. Yang et al. [60] describe a set of design benchmarks to measure the effectiveness of the static graphical representation of the VPE. These include indicators of the visibility of the VPE logic, visibility of dependencies, and use of screen real estate. Though these benchmarks do not specifically addresses all the issues in visualization user interface design, they can be applied to measure or improve the effectiveness of the user interface.

8.2 Classification of Visualization User Interfaces

User interfaces for visualization can be classified based upon the interface's display of the visualization exploration process—the process of generating visual results by navigating through visualization parameter space. This space consists of all the possible values for the parameters required by the underlying techniques; in the case of volume rendering,

these would be transfer functions, view position, lighting parameters, etc.

8.2.1 Interactive Control & Dynamic Manipulation Interfaces

The first two interface types discussed are interactive parameter control and dynamic manipulation interfaces [46]. In the former interface, interactive manipulation of the parameter values does not correspond to interactive updates to the rendered result; in the later interface, the result is rendered interactively during parameter changes. Visualization transform editing—the process of creating visualization transforms through means such as data-flow networks—is not supported in these interfaces, though a fixed number of different transforms may be available for use. In these systems, the major action is the editing of parameter values (potentially from different parameter types in the case of interactive parameter control interfaces) to generate a visualization result; in dynamic manipulation interfaces, parameter values can vary over a continuous range during manipulation. 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 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.

8.2.2 Data-flow Interfaces

Data-flow interfaces utilize the data-flow visualization transform model. They present the data exploration process as a directed network of connected components which act upon the data sets or output of other stages 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 [1, 56, 61]. Depending on the system, parameter changes and rendering may be synchronous like dynamic manipulation interfaces or

asynchronous like interactive parameter control interfaces.

Data-flow interfaces provide better state display than traditional interface through the data-flow network; the network clearly displays the flow of information whereas the previous interfaces have 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 previous interfaces, 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.

8.2.3 Parameter-based Interfaces

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 have already been mentioned—the Design Galleries system [41] and Image Graphs [39]. The properties are these interfaces are now explored more in-depth.

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 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 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 desired. 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 navigate the entire graph [27]. 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.

8.2.4 Spreadsheet Interfaces

Spreadsheets are a subset of form visual programming languages where the user programs the contents of cells aligned in a grid; they are most familiar from applications such as Lotus 1-2-3 or Microsoft Excel. Spreadsheets for visualization can be characterized by the following properties:

- A tabular structure that encapsulates the visualization process.
- Operators which act upon the contents of the spreadsheet to generate new spreadsheet cell values.
- Relationships between cell values that are either statically or dynamically updated.

Several spreadsheet interfaces for graphics and visualization have been developed. Levoy's SI system [38] 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 [26, 45]. Chi et al. [13] demonstrate a set of principles for visualization spreadsheets through their SIV system. Each presents a tabular interface where a cell can hold arbitrary image data; thus, a cell displaying the results from a IR satellite could be side-by-side with one displaying meteorological information for the same observed region. This side-by-side comparison is one of the strengths of a spreadsheet over the previously mentioned representations. In addition, each of these spreadsheets systems have dynamic formulas for cells; changes to images or formulas propagate spontaneously to any descendant cells.

Spreadsheets overcome several of the weaknesses of the previous user interfaces. First, they allow easy comparison of different results, even across different data sets (unlike the image graph, for example). Like the parameter-based representations, they present an overview of the data exploration process at a glance. Though the organization of spreadsheet representations scales better than parameter-based interfaces, zooming and navigation can become difficult as the number and size of images increase. Finally, previous results are usually kept available and displayed; the exception is that in formula-based spreadsheets, modifying a cell formula replaces its previous result and any results dependent on that cell.

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 are addressed by the spreadsheet-like interface described in Chapter 9.

8.3 Desired Visualization User Interface Properties

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. From the previous sections, a set of desired visualization user interface properties for the data exploration process can be developed:

- *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 allow the user to maximize the re-use of previous results. The first requirement is obvious and satisfies the first and third of Springmeyer's criteria (generating and querying data) for scientific visualization interfaces listed in Chapter 2. The second property clearly identifies what steps were required to generate the visualization; this identification becomes important if the generation required several computationally expensive steps. It also fulfills other requirements in Springmeyer's criteria (examining, navigating, comparing, and classifying data). The third property leverages previous results in producing related visualizations and is also related to Springmeyer's data generation task. Finally, the last property assists in the documentation and reproducibility of the visualization session.

Interactive parameter control and dynamic manipulation interfaces only satisfy the first criterion; they are suited for quick prototyping of the underlying visualization technique, not for actual application. Data-flow architectures partially support all of the

properties. The flow network employed by these interfaces does give some indication of the relationship between parameters and a history of that particular visualization while the nodes of the flow network are operators themselves. The flow network does not display the relationships between different visualization results nor does it keep track of changes in the network after modifications. Image graphs do satisfy these conditions; but, as parameter-based representations, they do not display the technique used as well as a data-flow interface. This becomes important in situations where several visualization techniques, with varying types of parameters, can be applied. In addition, the history information captured by the Image Graph can only be used in other Image Graphs; it does not use a general and complete model of the visualization process like the P-set Model.

The contrast between data-flow and parameter-based interfaces such as the image graph must be highlighted. Data-flow excels at displaying control flow (through the flow network); parameter-based representations cleanly identify the parameters being used. One of the reasons data-flow interfaces are popular is the ease with which new visualization techniques can be added—all that is needed is a module implementing the method; parameter-based interfaces focus on displaying the parameter values for a single technique. The parameter interfaces posses operators to create visualizations from previous results; data-flow networks have no similar operations besides direct manipulation. An user interface combining these two approaches would join their strengths and minimize their weaknesses: i.e., a data-flow network can be used for controlling the visualization process while an image graph would display the history of the network or be used for navigation. This integration is beyond the scope of this work. Instead, the principles for interfaces discussed will be implemented in a spreadsheet-like interface that utilized the P-set Model discussed previously. This interface is discussed next.

Chapter 9

A Spreadsheet-like Interface for Visualization Exploration

In order to gain insight from large 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 called the VisSheet that satisfies these requirements has been developed. This chapter discusses its capabilities and illustrates its uses through a series of examples.

Conventional spreadsheets have three properties [13]: 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. The VisSheet 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. The VisSheet builds upon the strengths of spreadsheets while augmenting them towards the task of visualization exploration.

Besides presenting an interface to the visualization process, the VisSheet also captures that process for the user using the P-set Model for visualization exploration. 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 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.

9.1 Spreadsheet-based Visualization Representation

Before the discussion of the structure of the spreadsheet-like interface, the conceptual model behind it must be understood. As suggested in Spence [51], 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 the internal model describes the properties of the interface.

9.1.1 Conceptual Model

Using the Visualization space model (Section 4.3.1), the VisSheet considers visualization exploration a process of examining a multi-dimensional space $\mathcal{V}$ of parameter values. Each n-tuple $p \in \mathcal{V}$ represents a combination of parameters that produce a visualization—a p-set. The p-set p uniquely identifies a point in the visualization space which corresponds

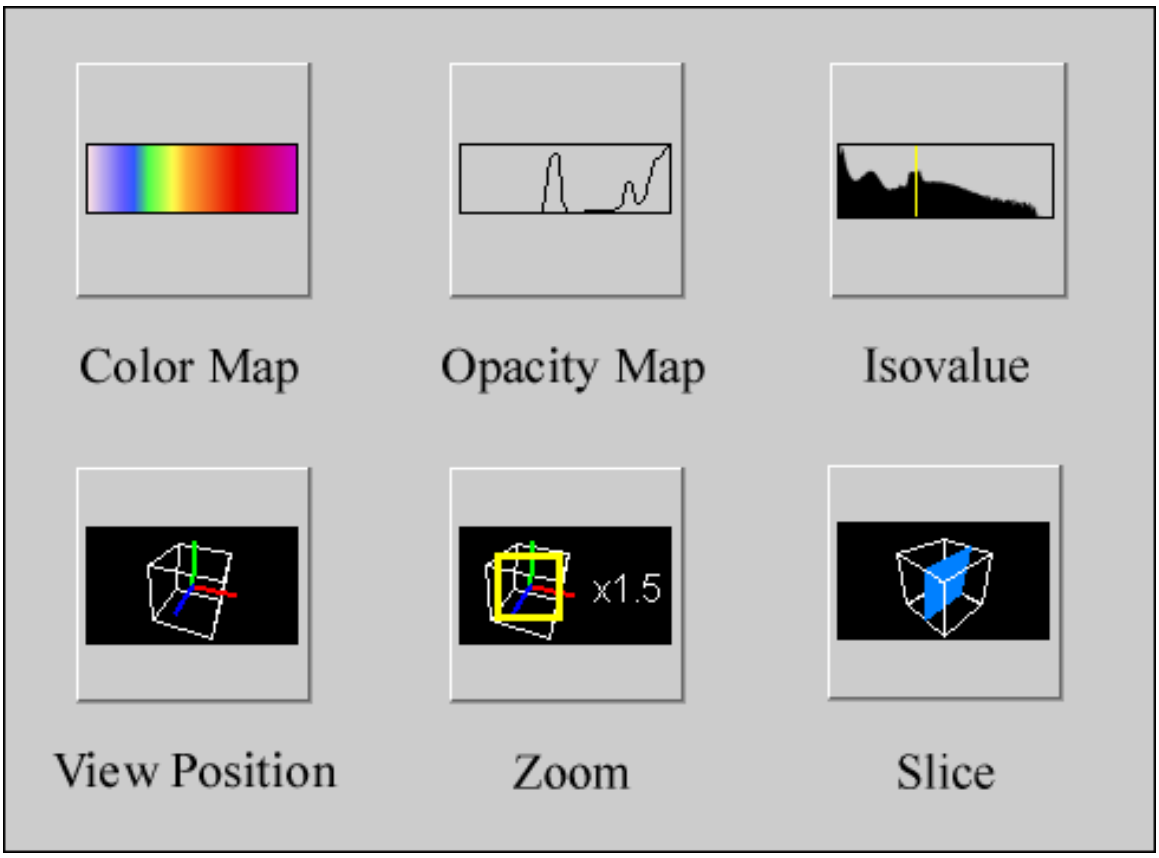


Figure 9.1: Visual representation of some parameters displayed by the visualization spreadsheet. As a user edits the underlying parameter, the icon of the parameter is updated.

to a particular visualization result r in some result space R. $\mathcal{V}$ thus defines a transformation $\mathtt{t} : \mathcal{V} \mapsto R$ from the parameters to the results. This transformation is the underlying visualization technique used and is described by some visualization transform model. 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 the 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, the p-sets and the derivations of those p-sets are captured by the spreadsheet. In addition, the results are also stored. Thus, any time a user generates a new row or column by any means—by parameter editing, operator, or by using the interpreter as discussed later—the new results are recording by the visualization session model. This information is also used in displaying history information, as discussed later in this chapter.

9.1.2 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. The VisSheet 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-time. Parameter values are represented by rendered glyphs in the table headers (Figure 9.1). 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 default values for non-displayed parameters, the spreadsheet "window" can be translated in visualization space (Figure 9.2). A different kind of motion is achieved by changing the displayed row and column parameters (Figure 9.3). Thus, the data exploration process becomes the process of manipulating the spreadsheet window through the visualization space.

Previous visualization spreadsheets collapse the 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. This structure satisfies the second principle of visualization user interfaces (Intuitive Display).

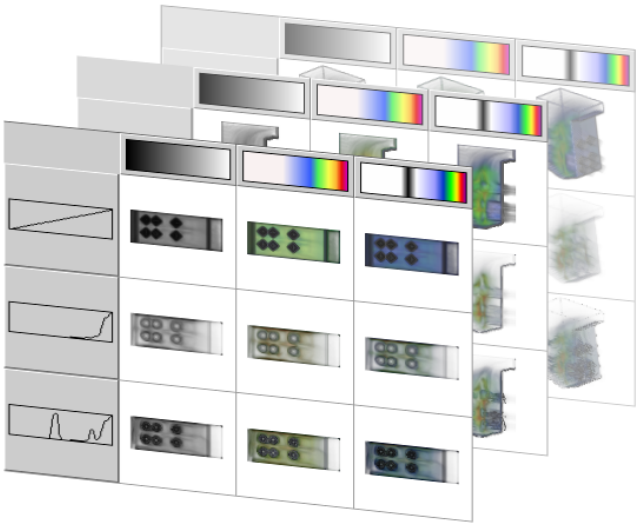


Figure 9.2: The VisSheet is a view of two dimensions of a visualization space. In this example, opacity maps are displayed along rows and color maps along columns. A particular cell is rendered by combining the non-displayed parameters' default values with the parameter values corresponding to the row and column indices. By changing the default parameters, in this case the view position, the spreadsheet's position in visualization space can be moved.

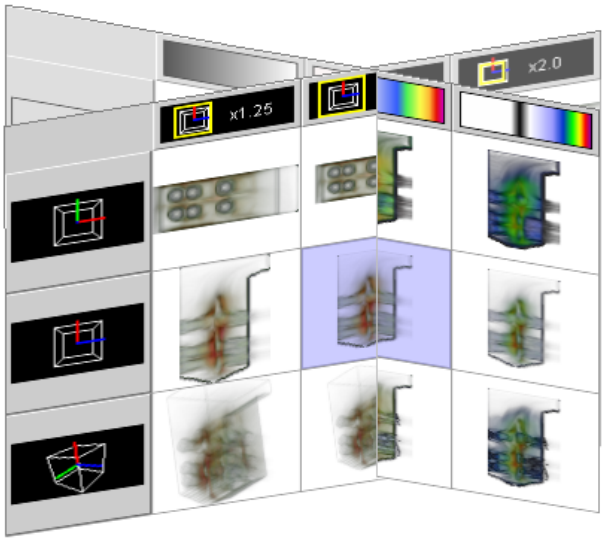


Figure 9.3: The spreadsheet window can also be rotated in visualization space. Starting from a sheet displaying color and opacity maps, the user first selects an image with the desired properties. These two parameters will become the new default values. By then selecting two new parameters to display, in this case view position and zoom factor, the window is rotated about the selected point to display the new values.

9.2 Static Spreadsheet-based Exploration

Without the dynamic operators and the interpreter described later, the 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. This mode of exploration fulfills the first principle of a visualization user interface (Parameter Manipulation). 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 9.4 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

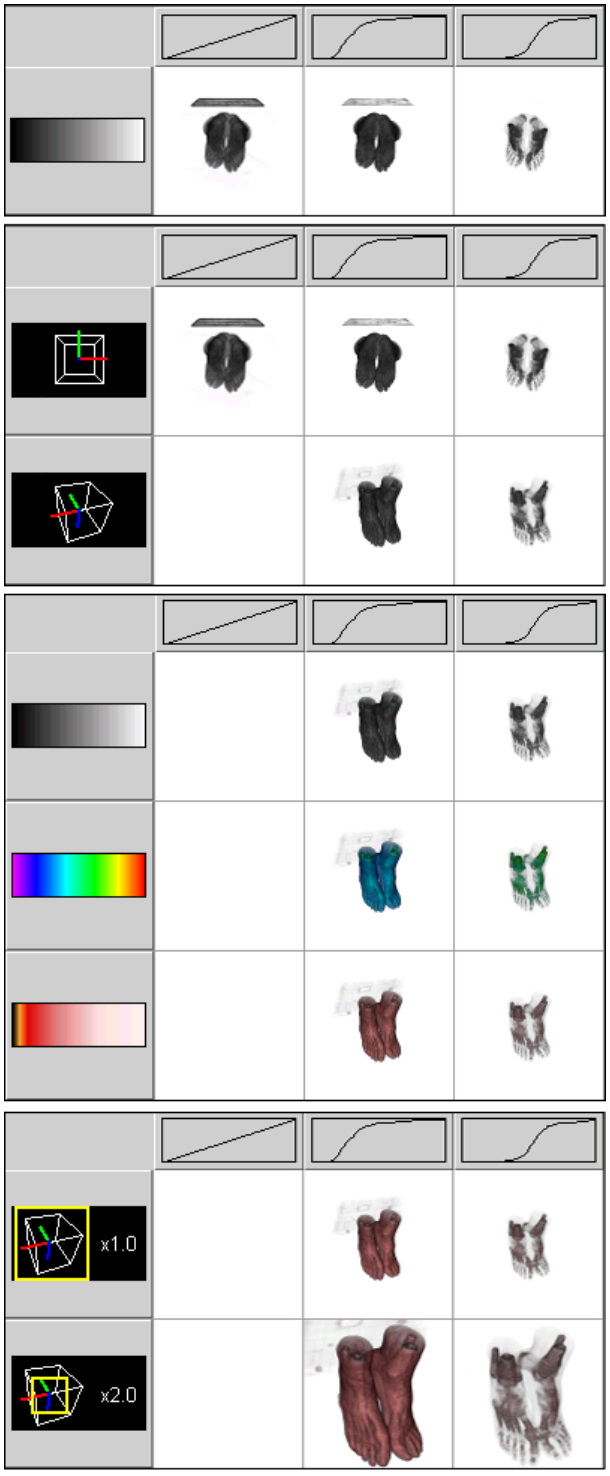


Figure 9.4: A sequence of spreadsheets displaying the volume visualization of a foot data set. Blank cells represent non-rendered images. The goal was to compare skin and bone surfaces. The user first determined appropriate opacity maps before modifying the view position, color map and zoom factor. The spreadsheet was useful in displaying the images to be compared side-by-side.

a new zoom factor value. If desired, the user 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. Figure 9.5 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.

9.3 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. Combined with the interpreter (c.f.), the dynamic operations satisfy the third principle of visualization user interfaces (Visualization Operators).

9.3.1 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. The VisSheet 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 are 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

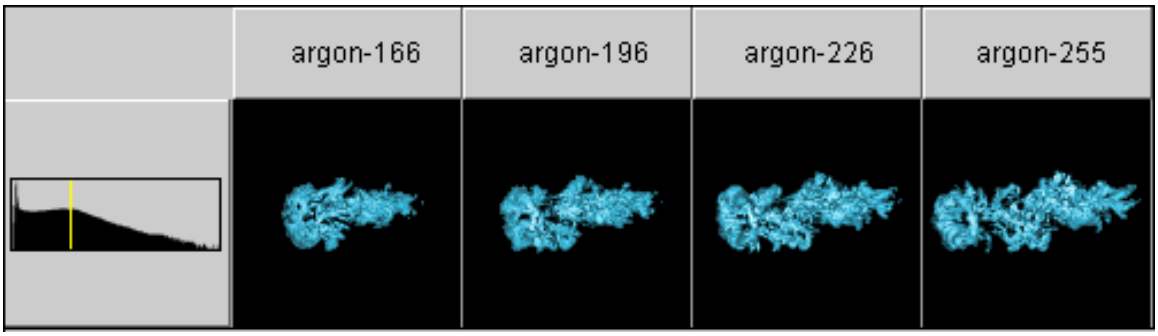


Figure 9.5: An isosurface visualization spreadsheet displaying the effects of a shock wave on a bubble of argon over several time steps. Modifying the displayed isovalue would change all the cells at once, a task that would be more difficult in other representations.

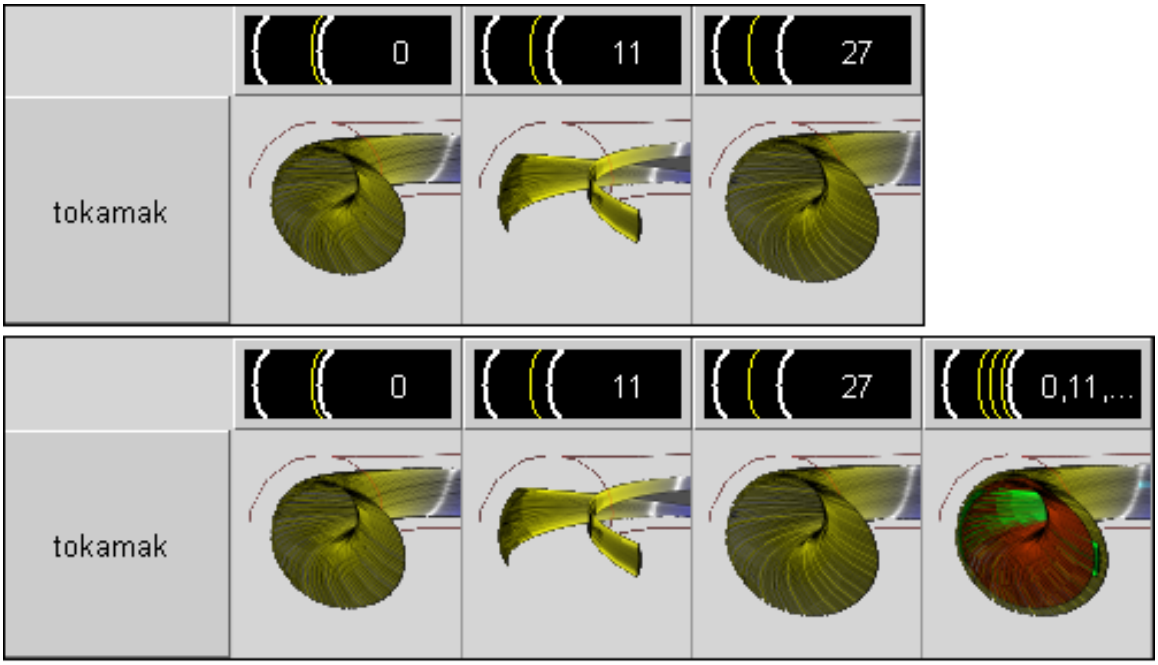


Figure 9.6: An example of applying parameter operators, in this case the union of displayed magnetic field lines in a simulation. The top image displays the three different field lines side by side. When combined, the presence of the "magnetic island" between the layers becomes apparent.

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 9.6 displays the illuminated magnetic field lines in a Tokamak simulation [49]. 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 9.7), 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 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.

9.3.2 Animations

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

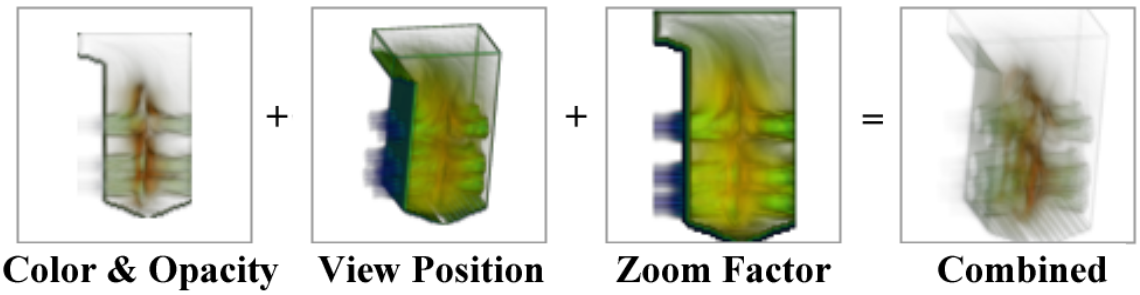


Figure 9.7: Selected cells for a composition operator. The user wished to use the color and opacity maps of the first image, the view position of the second and the zoom factor of the third. The fourth image displays the desired visualization.

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 9.7. 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 of the parameters. The corresponding animation would illustrate the 3-dimensional features of the flow through the furnace.

9.3.3 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 [43]. 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.

The VisSheet's implementation of scripting differs from common macro languages in numerical spreadsheets or the scripting abilities in the spreadsheets described by Chi et al. [13]. 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 the VisSheet, 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 this interface, references to a cell are translated into the tuple 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 9.8). 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 [59] 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 9.9 displays the results of executing this script. These scripts are a versatile way to drive the visualization process.

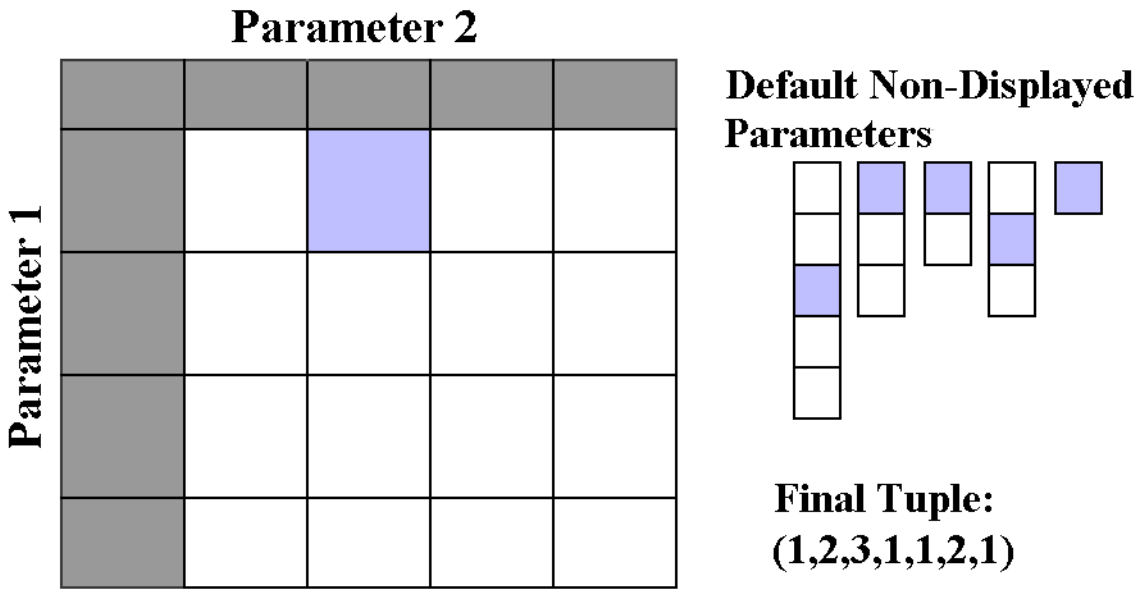


Figure 9.8: An example of referencing a cell. A reference to a cell, in this case cell `(1,2)`, is translated into a tuple representing the positions of the cell's parameter values in their respective parameter lists. The translated reference is `(1,2,3,1,1,2,1)`.

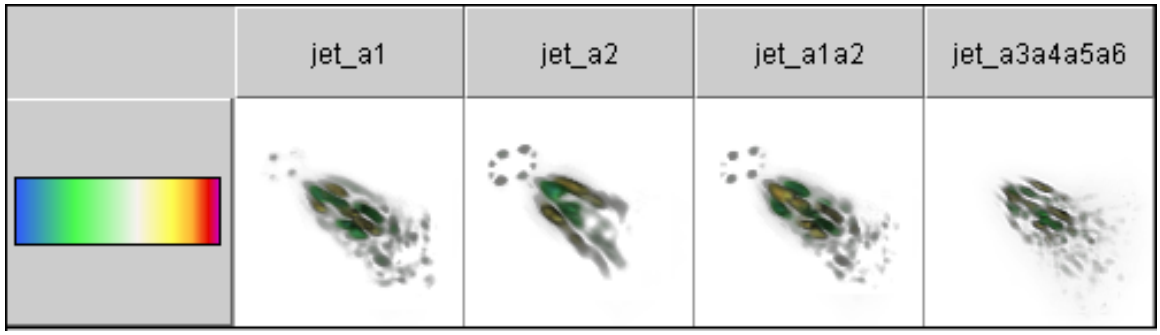


Figure 9.9: Another spreadsheet examining multiple data sets. The data represent distinct variables in a multi-variate turbulent jet simulation. The entire simulation has 9 variables. Two of the variables are displayed in the first two columns. The third column is the sum of the first two variables over 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.

9.4 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.

9.4.1 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, an alternative view of the process could be constructed. For example, by using the internal model discussed in Part II, any of the visualization process graphs could be displayed. Also, an HTML page summarizing the session (as mentioned in Chapter 7) could be generated.

9.4.2 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. Either of these options can be achieved using the framework discussed in Chapter 10.

9.4.3 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. All of these goals are accomplished by using the XML representation discussed in Chapter 7.

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.

9.5 Further Examples

To illustrate the versatility of the spreadsheet-like interface, two further examples are presented. 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 nine parameters (see [53] 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 VisSheet 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 9.10). 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 9.11. 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

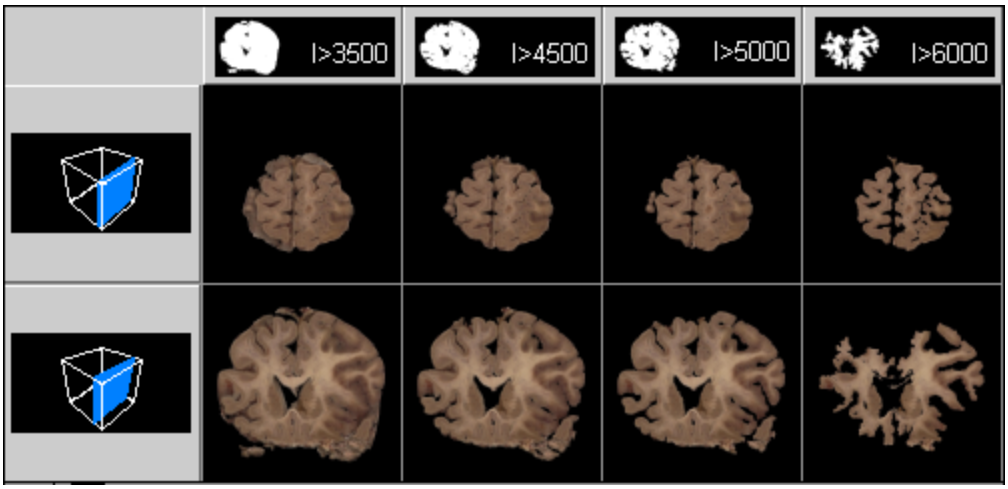


Figure 9.10: A spreadsheet analyzing the effect of different parameters on a 3D segmentation pipeline of a human brain. In this example, the effect of color thresholding (the column parameter) is examined across two different slices of the brain (the row parameter).

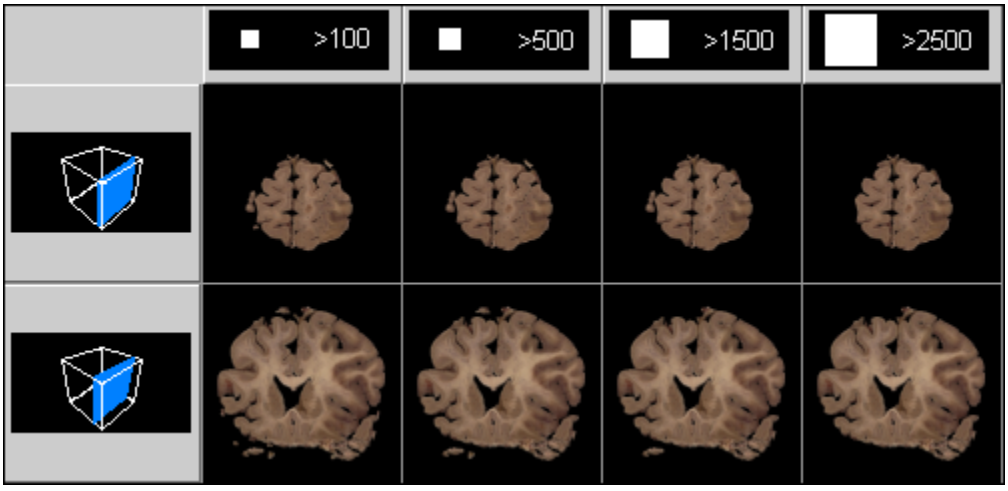


Figure 9.11: Another spreadsheet examining the segmentation pipeline. This time, different region size thresholds are displayed along the columns. Note how the effects of the region growing threshold parameter are less pronounced than those from the previous color thresholding stage.

during the segmentation 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 9.10 (a non-displayed parameter in Figure 9.11), 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 VisSheet with another application. In this configuration, 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. If the interactive volume renderer also implements the P-Set Model for its internal representation, then the communication of state is automatic via the framework discussed in the next chapter.

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 9.12. 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.

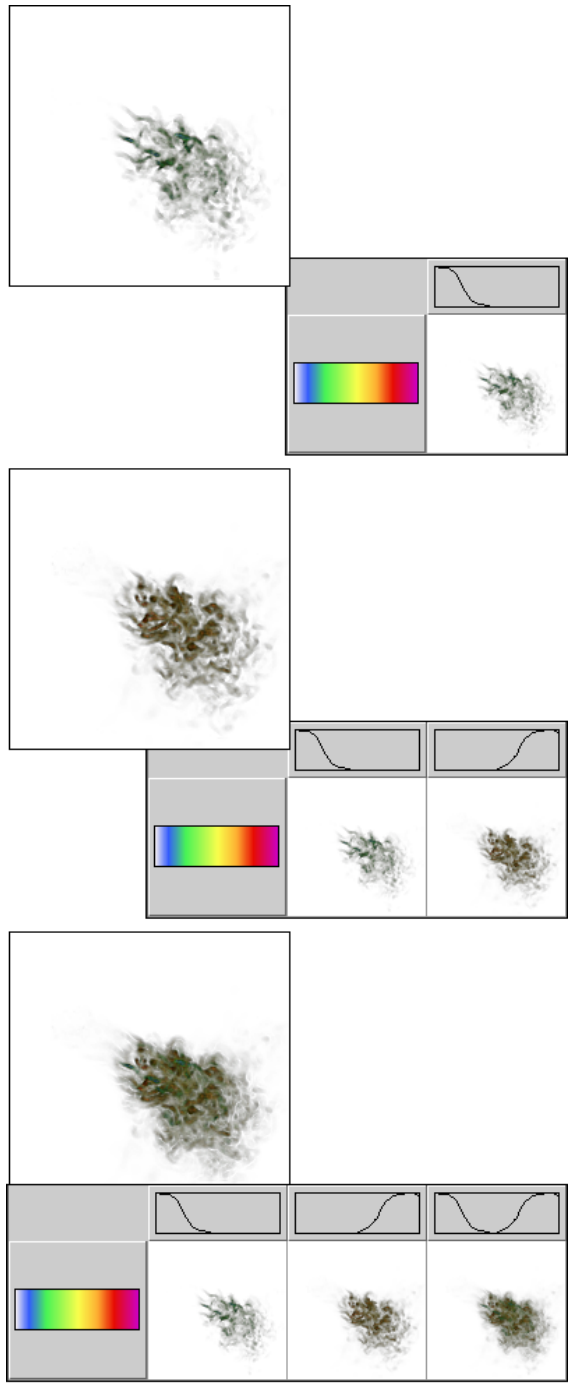


Figure 9.12: The spreadsheet can be coupled with an auxiliary program that can control the visualization. As the exploration progresses, the main application updates the spreadsheet with its current state. This interactive history mechanism can also be used to communicate the other way. Here the user applied a parameter operator to the opacity maps (columns) to highlight both the negative and positive vorticities in a turbulent jet data set.

9.6 Summary

By visually organizing the data exploration process while providing tools to build upon and share this process, the VisSheet 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.

Part IV

Summary

Chapter 10

A Framework for Visualization Exploration

In this chapter, the internal and external representation are brought together with a framework. This framework provides a set of classes for a complete visualization exploration system implementing the P-set Model and following the visualization exploration user interface principles. The uses of the complete framework are then demonstrated via an example.

10.1 The Core Framework

The core framework which implements the internal representation of the visualization process consists of the classes in Table 10.1. In the framework, there is a distinction between concrete classes (sans-serif type) and abstract classes which must be specialized (*italic sans-serif type*); member variables and functions are in `type-writer type`. The main functions of each component and its interactions with the other components are discussed below. Figures 10.1–10.2 provide UML class diagrams [3] which detail the dependence of the classes upon each other.

Component	Purpose
VisualizationSession	Stores visualization session results. Updates all views when a new result is generated.
VisualizationSessionResult	Represents a single visualization result.
Derivation	Represents information describing how a parameter was derived using the parameter calculus.
VisualizationTransform	Describes how the visualization is performed. Possesses a signature identifying the parameter and results type used in the transformation. Also responsible for rendering results.
VisualizationParameterType	Parameter type for a visualization transform, e.g. a colormap.
VisualizationParameterValue	Particular value for a parameter, e.g. a rainbow colormap.
VisualizationResultType	Class representing the actual type of a result, e.g. a raster image or geometry.
VisualizationResultValue	Particular value for a result, e.g. an empty image.
VisualizationOperator	An operator upon parameters or results that generates other parameters or results.
VisualizationView	A UI and UI toolkit-independent class representing behaviors common to all visualization UIs.

Table 10.1: A summary of the components in the visualization exploration and encapsulation framework.

10.1.1 The VisualizationSession Class

The VisualizationSession class encapsulates the entire visualization session. As such, it contains information about the visualization transforms used during the session (via the `transforms` function). If a new visualization transform is used during the session, it can be added via the `addTransform` function. All the parameters used during the exploration can be accessed via the `parameters` function; `results` is used to access the map of p-sets to visualization results. Session results, which include timestamp and derivation information, are accessed via the `history` function.

In addition to storing information about the session, the VisualizationSession class also manages communication with the different visualization user interfaces (which implement the VisualizationView abstract class below). Views can be added or removed with the `attatchView` and `detachView` functions. All the views can be accessed via the `views`

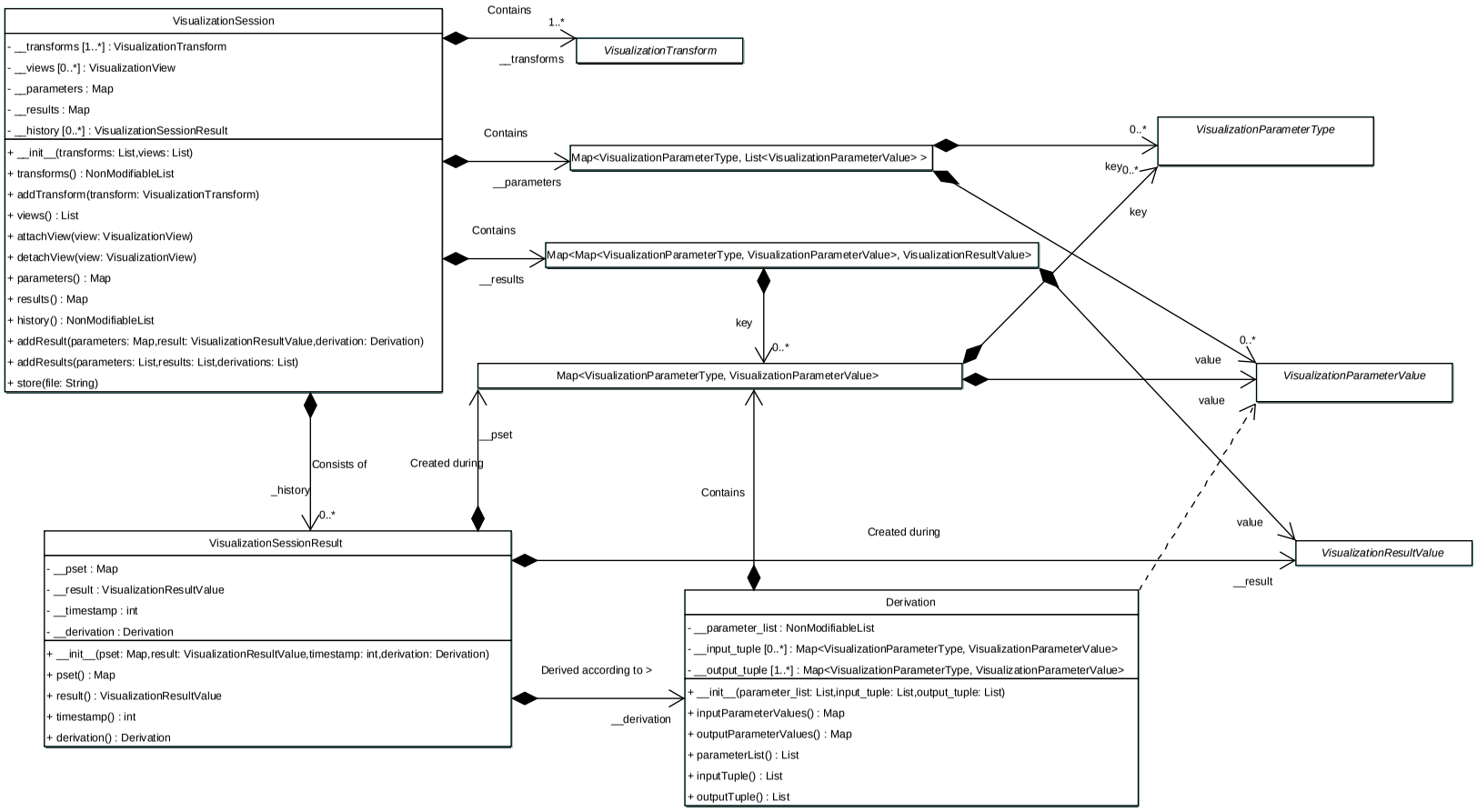


Figure 10.1: UML class diagram for the classes related to storing the visualization session. These classes encapsulate the user's actions during the visualization process.

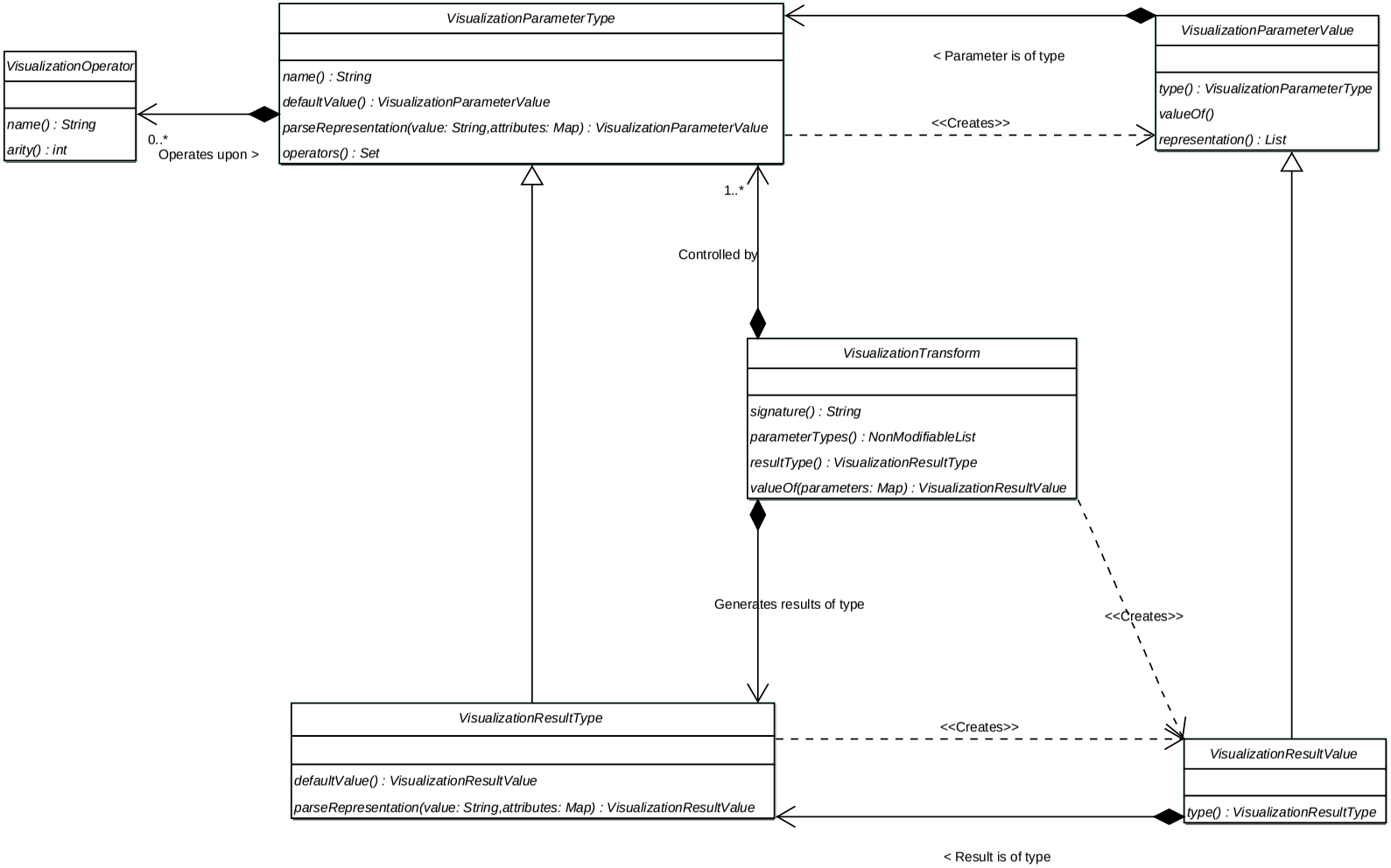


Figure 10.2: UML class diagram for the classes involved in the framework's type system. This type system describes the components of a visualization transform.

function.

The most important service of the session class is that it handles the adding of new results to the session. Using the `addResult` or `addResults`, one or multiple new results can be added. Each function takes a rendered result, its corresponding p-set, and its derivation information (or a sequence of those when adding more than one result). The results are then validated (to make sure they were created using a transform associated with this session) and a new session result is added to the session. In addition, any attached views are notified that the session has changed. Thus, if two views of the session are active, if one creates a new result, the other can display this new result concurrently.

Only one VisualizationSession object should be instantiated at a time. When a session is complete, it can be serialized using the XML representation by calling the `store` function. This function uses member functions of the result and parameter types to encode them along with the other information stored in the session. The `load` helper function can then be used to re-instantiate the session at a later time.

10.1.2 The VisualizationSessionResult Class

This class encapsulates a visualization session result: a p-set, rendered result, timestamp, and derivation information. Each can be accessed by their eponymously named functions. Only a VisualizationSession object can create session results; they are created automatically when a new result is added.

10.1.3 The Derivation Class

The Derivation class describes a p-set derivation during the visualization session. Factory functions for common derivation types (a parameter range or parameter application derivation) exist to create new Derivation instances. For more complex derivations, the object can be created directly by providing a list of parameter transformations, input p-sets, and output p-sets. All of these elements can be extracted from a Derivation instance using the `parameterList`, `inputTuple`, and `outputTuple` functions respectively.

10.1.4 The VisualizationTransform Abstract Class

The *VisualizationTransform* abstract class describes the visualization type being performed and handles the actual generation of rendered results. It is part of a type system describing the components of a visualization transforms—parameter and result types and values. The `parameterTypes` and `resultType` functions return the parameter and results type classes used by the transform; the `signature` function returns a string summarizes these types. More importantly, to actually render a result, the transform's `valueOf` function must be called. This function takes a p-set and returns a rendered result. If one of the parameters is undefined, an undefined result is also returned. Since the transform has no information about how the result was rendered (the user interface has this information), it does not handle adding results to the system. As mentioned, the view must add the result by providing the derivation information the session's `addResult` function.

When implementing a new visualization transform, not only does a *VisualizationTransform* class have to be implemented, but the various parameter and result type and value classes discussed below must be implemented. In addition, for the views, parameter and result renderers and editors will also have to implemented. These latter classes, however, are more specific to a specific view and thus are not part of the framework.

10.1.5 The VisualizationParameterType and VisualizationResultType Abstract Classes

These two abstract classes represent the parameter and visualization result types respectively. Each parameter and result type has a name (accessed through the `name` function), a default value (accessed through the `defaultValue` function), and a list of applicable visualization operators (accessed through the `operators` function). The default value is used when generating a default p-set for a visualization transform; in cases where there is no appropriate default (such as for data sets), an undefined value is used.

Implementations of these classes are also used during session loading from a stored representation. The `parseRepresentation` function uses the parsed element data and XML attributes for the parameter or result and instantiates an appropriate class instance. It is

the reverse of the `representation` function provided by the value abstract classes below.

10.1.6 The VisualizationParameterValue and VisualizationResultValue Abstract Classes

Like the type abstract classes, these abstract classes encapsulate information about parameters and results. In this case, they encapsulate parameter and result values. The type class of the value can be accessed via the `type` function and the actual value is accessed through the `valueOf` function.

Specializations of these classes are used during process serialization. The `representation` function generates the textual data stored in the XML element and a mapping of attribute-values pairs for the element. This information is extracted by the specializations of the type classes during loading.

10.1.7 The VisualizationOperator Abstract Class

The *VisualizationOperator* abstract class describe visualization operators—functions that act on parameters or results to generate other parameters or results. As expressed in Chapter 8, these operators are important in structured visualization exploration. The class exposes to functions: `name` and `arity`. The former provides a name for the function and the later is a list of parameter or result types needed to perform the function. The actual operation is performed by treating specializations of the class as functors—function objects. By calling the object with the appropriate number of type of parameters, the operator is performed.

10.1.8 The VisualizationView Abstract Class

This abstract class represents a visualization user interface. It is an interface and UI-toolkit independent class—it describes the properties inherit in all visualization interfaces.

The *VisualizationView* provides access to both its internal visualization session (via the `session` function) and to its viewable visualization types (via the `transforms`

function). A visualization interface may not be able to display all the transforms currently in a session; thus, the `transforms` function only returns those transforms that are viewable. New transforms may be added via the `addTransform` function.

As mentioned previously, specializations of the view class are responsible for adding new results to the visualization session. This is required since only the view knows what type of interactions the user performed that generated the result. Thus, four steps occur when a user generates a result:

1. The result value is created using the current *VisualizationTransform*s' `valueOf` function using the p-set provided by the interface.

2. The derivation for the result is instantiated.

3. The result value, corresponding p-set, and derivation is passed to the session's `addResult` function, thus updating the session.

4. During the `addResult` call, each of the session's *VisualizationView* instances has its `update` function called to redraw themselves with the updated results.

When a new result is added, the session will call each view's `update` function, indicating that a new result was generated. It is then up to each view to update their display. These steps are illustrated for the VisSheet in the UML sequence diagram in Figure 10.3.

10.2 The VisSheet Framework

The VisSheet user interface prototype in Chapter 9 utilizes the core framework. It does so by specializing the *VisualizationView* interface to capture behavior specific to the VisSheet (see Table 10.2 and the UML class diagram in Figure 10.4).

10.2.1 The VisualizationSheetView Abstract Class

The base visualization sheet class extends the *VisualizationView* class in several ways. First, it provides access to its current displayed state via the `state` function; this returns the sheet's corresponding state object (see next section). One may also retrieve a cell's

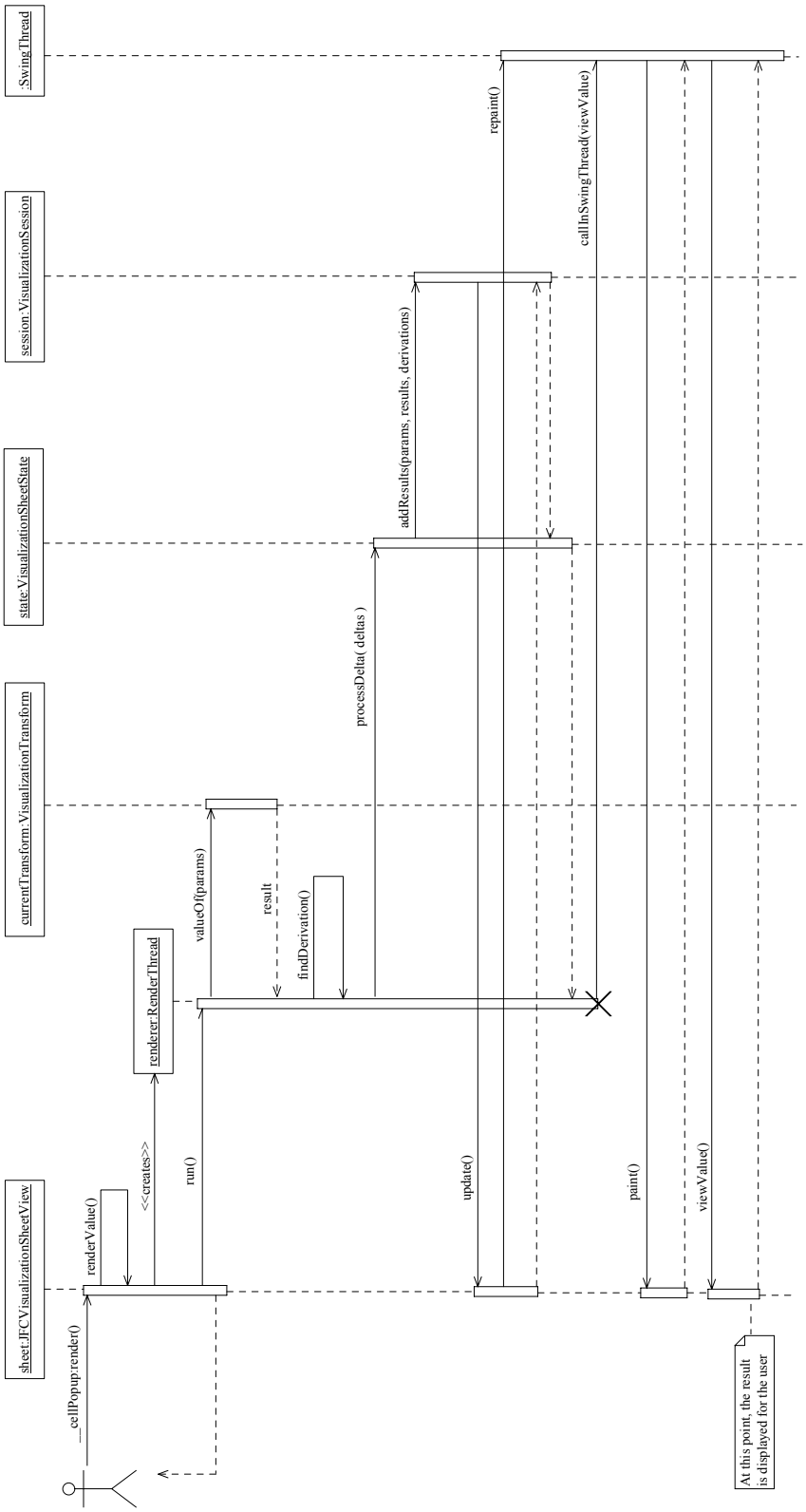


Figure 10.3: UML sequence diagram for adding and displaying a new visualization result using the VisSheet. The sequence follows four phases: Generating the result, determining its derivation, adding the result to the session, and updating the views of that session.

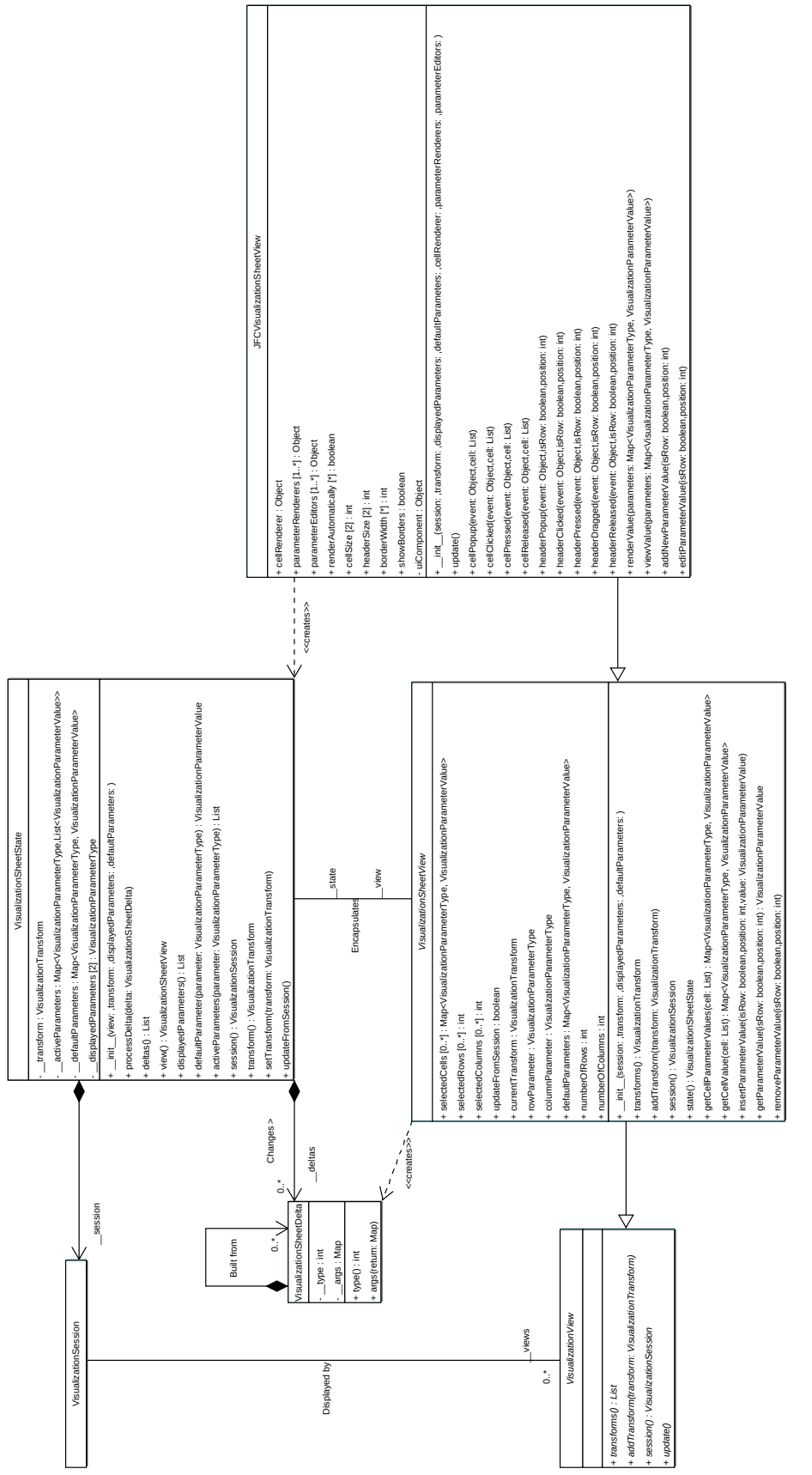


Figure 10.4: UML class diagram for the major components in the VisSheet and their dependence on components in the core framework.

Component	Purpose
VisualizationSheetView	The UI and UI toolkit-independent class describing the behavior of the VisSheet.
VisualizationSheetState	Encapsulates VisSheet specific state not shared by the visualization session, e.g. currently displayed rows and columns.
VisualizationSheetDelta	An atomic change in the state of the sheet, e.g., changing the displayed row and column.
JFCVisualizationSheetView	The Java UI class for the VisSheet.

Table 10.2: A summary of the components in the VisSheet framework.

parameter values or result (via `getCellParameterValues` and `getCellValue` respectively), retrieve a row or column's parameter value (via `getParameterValue`), and add or remove new rows or columns (via `insertParameterValue` and `removeParameterValue`). Several member variables also provide information about the sheet's displays, these are listed in Table 10.3. Changing the mutable variables often generates a sheet delta (c.f.) which updates the sheet state. Updates by the visualization session (via `update`) also generate sheet deltas.

When created, a *VisualizationSheetView* constructor must be passed the transform it will view, the visualization session it is a part of, and renderers and editors for each parameter and result type in the viewed transform. These are used whenever a user requests to add a new parameter (for the editors) or when displaying row, column, or cell values (for the renderers).

10.2.2 The VisualizationSheetState Class

The visualization sheet state captures non-session related state. This state includes which parameters are currently displayed (accessed via the `displayedParameters` function), the default parameters used (accessed via `defaultParameterValue`), and the parameters actually used by the sheet (accessed via `activeParameterValues`). The visualization session, the transforms, and the paired visualization sheet view can also be accessed via same-named functions. Finally, updates to the state of the sheet are performed by the `processDelta` function. These deltas, described in the next section, change the sheet's

Class	Purpose
`rowParameter`	The currently displayed row parameter. Mutable.
`columnParameter`	The currently displayed column parameter. Mutable.
`selectedRows`	The currently selected row indices. Mutable.
`selectedColumns`	The currently selected column indices. Mutable.
`selectedCells`	The currently selected cell p-sets. Mutable.
`defaultParameters`	The dictionary of default parameter values. Read-only.
`numberOfRows`	The number of currently displayed rows. Read-only.
`numberOfColumns`	The number of currently displayed columns. Read-only.
`updateFromSession`	Boolean value. If true (the default), changes to the visualization session from outside the sheet will be reflected by the sheet. Mutable.
`uiComponent`	The actual native UI component used to display the sheet. Read-only.

Table 10.3: Properties of the VisualizationSheetView abstract class.

state. Some of these delta's force a change in the session state by requesting a result to be rendered. In this case, the `processDelta` function will call the session's `addResult` function appropriately.

10.2.3 The VisualizationSheetDelta Class

The sheet state class stores process information in a manner 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, rendering a result, 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 interpreter—the atomic operations can be aggregated when needed. These deltas are encapsulated using VisualizationSheetDelta objects.

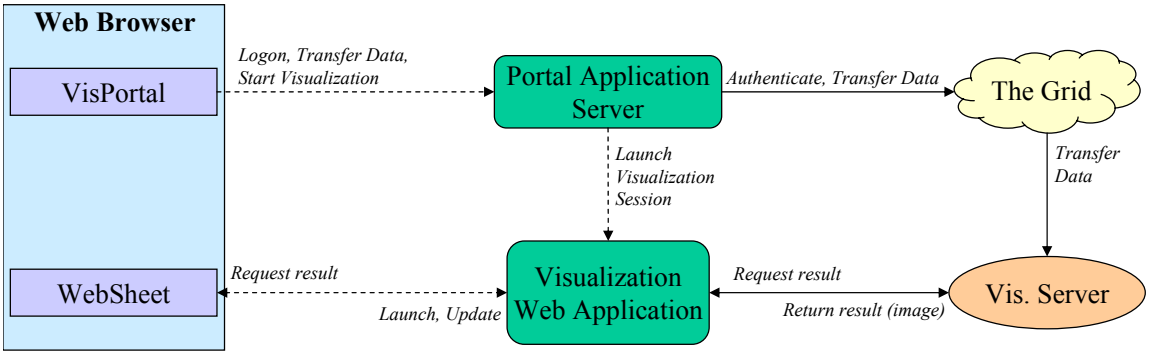


Figure 10.5: The VisPortal/WebSheet architecture. Elements within the blue box represent web pages that the user interacts with, green boxes represent web servers, and the other nodes represent grid-accessible resources. Dashed lines are HTTP/HTTPS connections for HTML and images (for the client) and connection requests (for the server), and solid lines are TCP connections. A line's label indicates the action performed over that link; for bidirectional connections, the top label corresponds to actions initiated from the left entity while the bottom label corresponds to actions initiated from the right entity. Note that the renderer can be located anywhere on the Grid—the VisPortal initializes the connection between the renderer and the visualization web application when a visualization is first requested.

10.2.4 The JFCVisualizationSheetView Class

This class implements the version of the VisSheet demonstrated in Chapter 9. The framework is implemented in Python [57], which is accessed via Jython (the successor to JPython [32]) bindings. The editors and renderers are also Jython classes. An alternate sheet implementation is detailed in the next section.

10.3 The Framework in Action

The VisSheet system is one example of the framework in action, though it does not take advantage of all the capabilities of the P-set Model. A much more complete usage of the framework is the VisPortal project conducted in collaboration with Lawrence Berkeley National Laboratory (LBNL). This section details this work, a web-based portal for visualization exploration and collaboration over the Grid.[1]

The LBNL VisPortal system consists of three major components: a web-based user interface to grid-enabled visualization services, a visualization web application which tracks

[1] "The Grid" is a distributed computing framework which handles user authentication, process control, data management and security [19].

the exploration of visualization results, and the portal application server that manages and coordinates the authentication for and use of grid resources (including the interface, web application, and volume renderer, see Figure 10.5). The application server (the VisPortal) uses established grid technologies to handle user and resource management (such as data transport). Once authenticated, a new visualization exploration session is initialized by the web application; the web application (also called a *servlet*) is a program on the web server that communicates with the client via HTTP (the Hypertext Transfer Protocol—the protocol for the World-Wide Web). In this case, the servlet maintains the visualization session state. After the visualization session is initialized, the web-based visualization interface is loaded in the client's web browser. As the visualization session progresses, the visualization results and the relationships between those results are stored by the web application for later examination. Finally, when the user is finished visualizing their data, the session is closed. The user can then initialize another session or re-examine previous explorations. The interplay between the interface and the web application is explained next.

10.3.1 A Web-based Sheet-like Interface for Visualization

The web interface implements an HTML version of the VisSheet interface discussed in Chapter 9. The sheet-like web interface shares many characteristics with the VisSheet. The interface refines the initial VisSheet design to allow a user to easily modify default and displayed parameters via the default parameter bar and drop-down row and column parameter lists (Figure 10.6). The default bar assists in the identification of parameter values and their corresponding results: The parameters are always the parameters belonging to a cell's row and column, combined with the default values for the other parameters in the default bar. Interaction with the tabular display remains essentially unchanged: users can add, edit, or remove parameter values; render or view a cell's image; and apply parameter and value operators to generate new rows, columns, or cells. The implementations of these two systems, however, differ significantly as the WebSheet uses JavaScript instead of Java for implementation. However, the JavaScript implements all the core functionality of the VisSheet framework discussed previously.

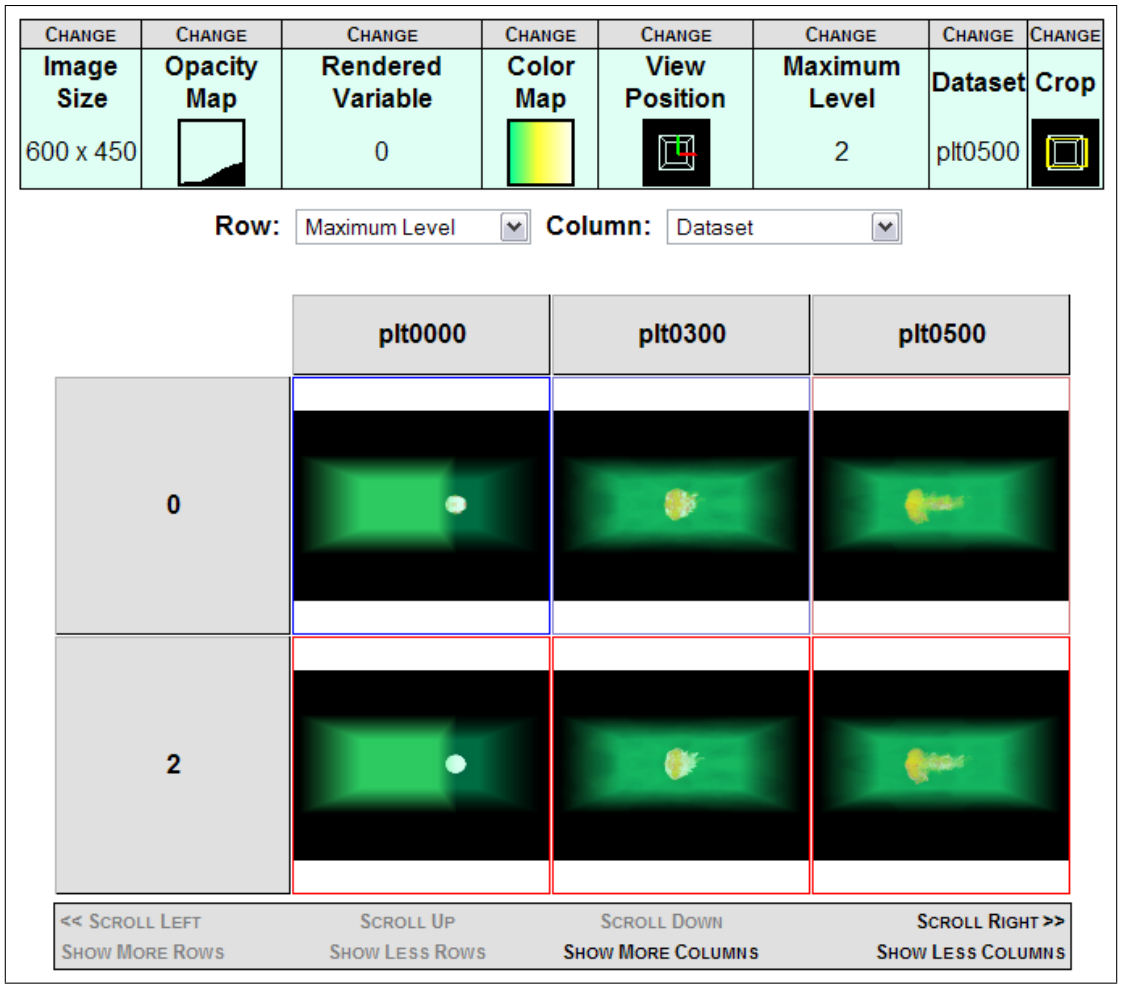


Figure 10.6: The AMRWebSheet interface, an example of the web interface to grid-based visualizations. The interface consists of three major areas: The default parameter bar that displays and allows the modification of the default parameter values; the displayed row and column parameter drop-down lists; and the tabular result display. The first two components are used to change the location of the tabular window in visualization parameter space while the last component is used to request the rendering of new results.

10.3.2 Web-based Encapsulation of Visualizations

The web-based visualization interface structures the visualization exploration process. The visualization web application server captures this process. By capturing the process, the system ensures that the visualization results generated, and the relationships between those results, are not lost when the visualization session ends. To record the visualization process, the P-set Model of visualization exploration is used. As each requested image is rendered, the corresponding visualization session result is stored by the web appli-

cation server. Thus, at the end of a session, all the rendered images, the parameter value sets (p-sets) used for creating that image, when that image was generated, and that image's relation to previous images are available for later use.

The visualization web application is the entry point to the web interface. When loaded from the portal, the servlet provides a user with two options: the user may start a new visualization session or view previous sessions. When the user chooses to start a new session, another servlet, the UI servlet, is loaded to handle interactions with the visualization UI. As the user requests images or adds, edits, or removes parameter values, the underlying JavaScript sends HTTP requests to this servlet. The servlet then processes the requests, contacting the rendering server if needed, and updates the visualization session and the client. The UI servlet represents the state of the UI; the interface's web page presents the view of this state.

If a user chooses to examine previous sessions, the session servlet is loaded. Initially, a list of all the previous explorations, sorted by date, is presented to the user. The list supports three actions. A user can re-load a previous session in the web interface by clicking on its corresponding link. New results can be added to this session; when the session terminates, these results will be stored along with the old session information. This capability is crucial to the VisPortal—scientists must be able to distribute their work over time as well as over space.

The second service the session servlet supports is the viewing of previous sessions. By selecting the "View as HTML" option, the user initiates the generation of an HTML page that summarizes the corresponding visualization session. Each result, the parameters corresponding to that result, and the parent and child results for that result are all part of the HTML page. The HTML page serves as an overview of a previous visualization session and as documentation of that session. First, the web page fully documents the visualization process as it completely describes the information captured by the visualization process model. Second, users are allowed to add or edit annotations of results. These annotations are stored on the web application server for others to access. Scientists can use these annotations to flag certain results as "interesting" to collaborators.

The session servlet also allows a user to view an overview graph of a visualization.

While the HTML session document describes the visualization in detail, it is difficult to obtain a sense of the visualization "at-a-glance." By selecting the "View Overview Graph" option from the session list, the servlet generates a graph depicting the results and various relationships between the results. The user chooses a visualization process graph to display (Figure 10.7. All the graphs use a new radial focus+context visualization technique [33]. In this technique, the radial distance from the center node to another node represents the distance of that node's result p-sets from the center result's p-set according to the chosen graph; as the distance increases, the size of the node and its radial separation decreases in order to allow the system to display all results simultaneously. Different process graphs and the HTML session view provide means of understanding what occurred during a visualization session.

The web application utilizes all the services provided by the framework. Sessions are stored using the internal representation. When a new session starts or is loaded, a new VisualizationSession object is instantiated with the appropriate visualization transform. The WebSheet, HTML overview, and process graphs each implement different *VisualizationView* servlet classes that manage the creation and interaction with the user's web browser. HTTP requests from the browser are processed by these same servlets in order to handle changes between the views, process information requests from the VisualizationSession object, or to generate new results. Similar servlet/JavaScript classes exist to perform parameter rendering and editing. When a user exists the portal, the VisualizationSession serializes itself into its XML representation that can be accessed again at a later date. Without the framework, these services would be more difficult to implement and integrate.

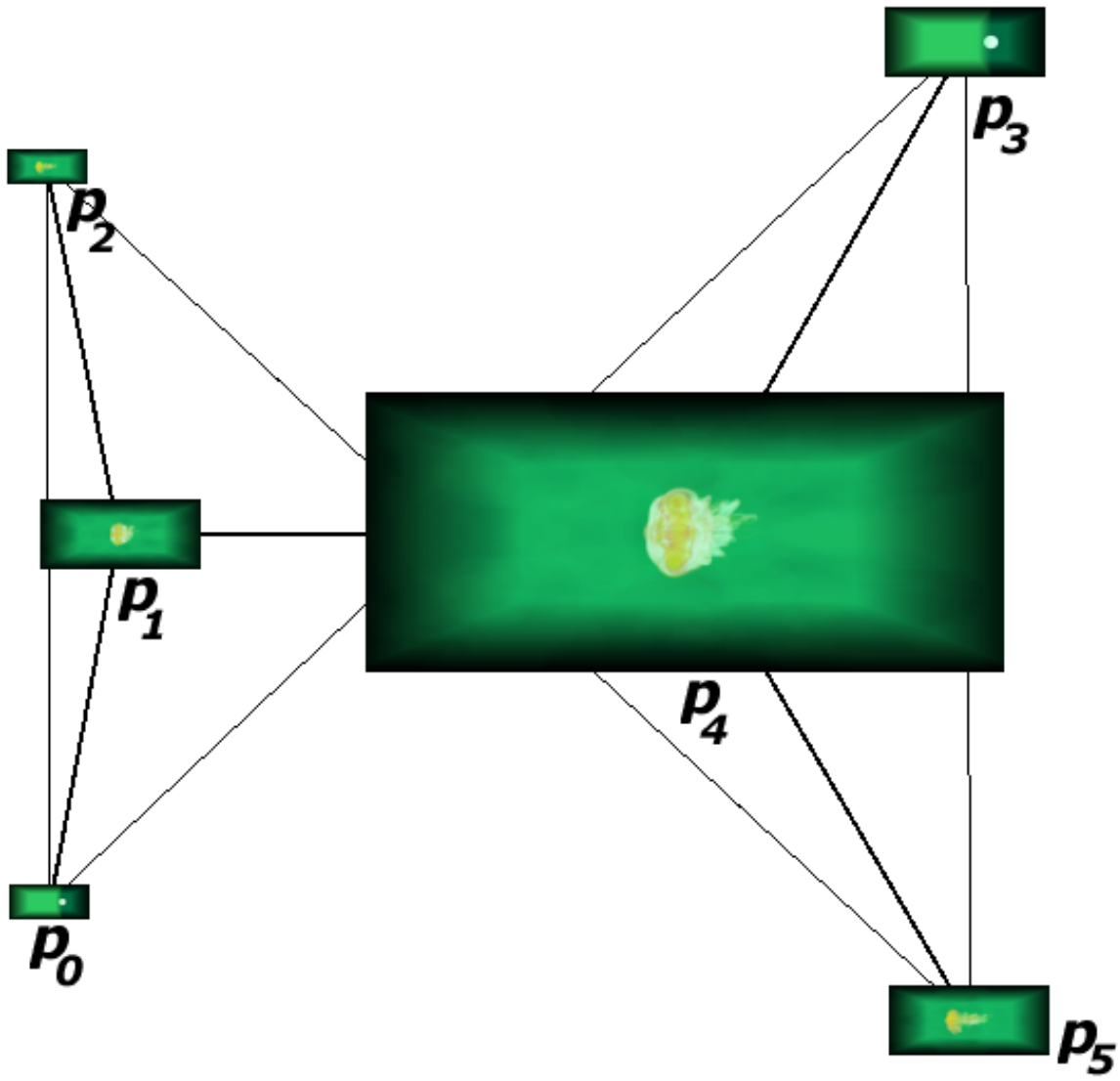


Figure 10.7: P-set difference visualization process graph for the session in Figure 10.6 using a focus+context layout. Edges indicate that only one parameter value differs between the two resulting images.

Chapter 11

Conclusions

The visualization exploration process contains a wealth of information; this work has demonstrated a model to describe this information and a representation to share the information. Both the visualization technique performed and the process used to generate visualization results are captured by the model and representation. In addition, principles for visualization exploration user interfaces have been developed to effectively utilize this model; the VisSheet demonstrates these principles in action. By visually organizing the data exploration process while providing tools to build upon and share this process, the spreadsheet-like interface makes visualization more efficient and effective. The space of visualization parameters is made clear by the VisSheet'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.

11.1 Effectiveness

To demonstrate the effectiveness of this research, the example from the introduction is revisited. This time, Alice and Bob use the VisPortal system utilizing the developed framework (discussed in Chapter 10). In this scenario, Bob generates the data and places it on the Grid for Alice to visualize.

At the beginning of the session, Alice enters the VisPortal URL into her web browser. After logging onto the system, she uses the portal's access to the Grid to transfer the argon bubble data set from its original location to the visualization server. Alice then requests a new visualization session from the portal. The portal then transfers control to the visualization web application.

Upon initialization, the web application determines whether Alice desires to start a new visualization or view/expand an older session. In this scenario, she starts a new visualization session. After specifying an initial data set, the WebSheet page is loaded in Alice's browser, a few results already generated from the default parameter values the WebSheet uses. She then explores the data via the web-page interface until she is satisfied with the results. Because the WebSheet uses the principles discussed in Chapter 8, comparison between results is trivial. Eventually, Alice terminates the visualization session and exits the portal. When Alice exits, the visualization session is automatically recorded by the system using the XML representation from Chapter 7. Unlike Alice's initial scenario, no manual recording of the session is required.

At some later date, Bob wishes to verify the results generated during the visualization. Like Alice, Bob logs on to the VisPortal to access Alice's exploration; alternatively, Alice could have sent Bob the visualization session XML file as it completely describes the session. After using the portal to load Alice's session, Bob chooses to examine an overview graph of the visualization session similar to those from Chapter 6. After familiarizing himself with the visualization results, Bob loads the HTML summary of the session. Bob then annotates a few results of interest and exits the system. As with the original session, the visualization web application stores Bob's annotations automatically when he exits. Later, Alice can reload the session, view Bob's comments, and perhaps add some comments of her own. The framework developed here allows users of the portal to focus on exploring their data, not managing it.

11.2 Impact

The framework presented here impacts the user of visualization in several ways. Systems utilizing the process model assist in reuse since they clearly track the fundamental interactions a user has with the visualization process. Visualizations sessions employing this formalism can be used in heterogeneous visualization interface environments, enabling large-scale collaboration. The salient details of the visualization process are documented, allowing others to reproduce the process. Finally, others can use the formal model to operate upon or analyze their results in a rigorous manner. The VisSheet, in turn, structures visualization exploration. Using the interface, the user always knows where they have been, where they are, and perhaps where they should go in their exploration.

This work also contributes to the understanding of the visualization process. A characterization of user interactions with parameters during the visualization process has been performed. This characterization has led to the development of a parameter derivation calculus to describe the relationships between results created during a visualization session. Information stored using this calculus can be analyzed and further visualized to gain insight in the visualization process itself.

In the future, this research can be extended in several ways. Popular visualization user interfaces (such as existing data-flow interfaces) and toolkits should be extended in order to incorporate this research. Such extensions will allow these systems to benefit from various aspects of this work (e.g., visualization session dissemination). Alternate visualization user interfaces using this framework can also be envisioned—consider, for example, a focus+context version of an Image Graph coupled with an interactive interpreter like the VisSheet. Finally, the P-set Model only captures *what* occurred during a visualization session; a meta-data model built upon the P-set model would capture *why* an operation was performed. With this research as a base, visualization collaboration will become easy and effective.

Bibliography

[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] Jacques Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. University of Wisconsin Press, 1967/1983.

[3] Grady Booch, Ivar Jacobson, and Jim Rumbaugh. OMG United Modeling Language specification (version 1.5). Technical Report formal/03-03-01, Object Management Group, March 2003.

[4] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). Recommendation, World Wide Web Consortium, 2000. http://www.w3.org/TR/REC-xml.

[5] Ken Brodlie, Andrew Poon, Helen Wright, Lesly Brankin, Greg Banecki, and Alan Gay. GRASPARC–A problem solving environment integrating computation and visualization. In Gregory M. Nielson and R. Daniel Bergeron, editors, *Proceedings of the IEEE Conference on Visualization 1993 (Vis '93)*, pages 102–109. IEEE Computer Society Technical Committee on Computer Graphics, IEEE Computer Society Press, October 25–29 1993.

[6] Margaret M. Burnett. Visual programming. In John G. Webster, editor, *Encyclopedia of Electrical and Electronics Engineering*. John Wiley and Sons Inc., New York, 1999.

[7] David M. Butler and M. H. Pendley. A visualization model based on the mathematics of fiber bundles. *Computers in Physics*, 3(5), September/October 1989.

[8] Stuart K. Card and Jock Mackinlay. The structure of the information visualization design space. In John Dill and Nahum Gershon, editors, *Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis '97)*, pages 92–99, 125. IEEE Computer Society Technical Committee on Computer Graphics, IEEE Computer Society Press, October 20–21 1997.

[9] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, 1999.

[10] Stephen M. Casner. Task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics*, 10(2):111–151, 1991.

[11] Ed H. Chi. Expressiveness of the data flow and data state models in visualization systems. In *Proceedings of the International Working Conference on Advanced Visual Interfaces 2002 (AVI '02)*, pages 375–378. ACM, ACM Press, 2002.

[12] Ed Huai-Hsin Chi. A taxonomy of visualization techniques using the data state reference model. In Steven F. Roth and Daniel A. Keim, editors, *Proceedings of the 2000 IEEE Symposium on Information Visualization (InfoVis '00)*, pages 69–75. IEEE Computer Society Technical Committee on Visualization and Graphics, IEEE Computer Society Press, October 9–10 2000.

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

[14] Ed Huai-Hsin Chi and John T. Riedl. An operator interaction framework for visualization systems. In John Dill and Graham Wills, editors, *Proceedings of the 1998 IEEE Symposium on Information Visualization (InfoVis '98)*, pages 63–70, 1998.

[15] Mei C. Chuah and Steven F. Roth. On the semantics of interactive visualizations. In Stuart Card, Steven Eick, and Nahum Gershon, editors, *Proceedings 1996 IEEE Symposium on Information Visualization (InfoVis '96)*, pages 29–36. IEEE Computer Socieity Technical Committee on Computer Graphics, IEEE Computer Society, October 28–29 1996.

[16] James Clark. XSL Transformations (XSLT) Version 1.0. Recommendation, World Wide Web Consortium, 1999. http://www.w3.org/TR/xslt.

[17] Gitta O. Domik and Bernd Gutkauf. User modeling for adaptive visualization systems. In R. Daniel Bergeron and Arie E. Kaufman, editors, *Proceedings of the IEEE Conference on Visualization 1994 (Vis '94)*, pages 217–223, CP 24. IEEE Computer Society Technical Commitee on Computer Graphics/ACM SIGGRAPH, IEEE Computer Society Press, October 17–21 1994.

[18] David Duke, Giorgio Faconti, Michael Harrison, and Fabio Paternò. Unifying views of interactors. In Tiziana Catarci, Maria F. Costabile, Stefano Levialdi, and Giuseppe Sanctucci, editors, *Proceedings of the Workshop on Advanced Visual Interfaces (AVI '94)*, pages 143–152. ACM SIGCHI, ACM Press, June 1–4 1994.

[19] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[20] N. Freed and N. Borenstein. Multipurpose Internet Mail Extentions Part Two: Media Types. Technical report, The Internet Engineering Task Force, 1996. RFC2046.

[21] Issei Fujishiro, Yoshihiko Ichikawa, Rika Furuhata, and Yuriko Takeshima. GADGET/IV: A taxonomic approach to semi-automatic design of information visualization applications. In Steven F. Roth and Daniel A. Keim, editors, *Proceedings of the 2000 IEEE Symposium on Information Visualization (InfoVis '00)*, pages 77–83. IEEE Computer Society Technical Committee on Visualization and Graphics, IEEE Computer Society Press, October 9–10 2000.

[22] Issei Fujishiro, Yuriko Takeshima, Yoshihiko Ichikawa, and Kyoko Nakamura. GADGET: Goal-oriented application design guidance for modular visualization environments. In Roni Yagel and Hans Hagen, editors, *Proceedings of the IEEE Conference on Visualization 1997 (Vis '97)*, pages 245–252,548. IEEE Computer Society Technical

Committe on Computer Graphics/ACM SIGGRAPH, IEEE Computer Society Press, October 19–October 24 1997.

[23] Robert B. Haber, Bruce Lucas, and Nancy Collins. A data model for scientific visualization with provisions for regular and irregular grids. In Gregory M. Nielson and Lawrence Rosenblum, editors, *Proceedings of the IEEE Conference on Visualization 1991 (Vis '91)*, pages 298–305, 1991.

[24] Robert B. Haber and David A. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. In Gregory M. Nielson, B. Shriver, and Lawrence Rosenblum, editors, *Visualization in Scientific Computing*. IEEE Computer Society Press, 1990.

[25] Richard Hamming. *Numerical Methods for Scientists and Engineers*. MacGraw-Hill, 1962.

[26] A. Frederick Hasler, Kannappan Palaniappan, and Michael Manyin. A high performance interactive image spreadsheet (IISS). *Computers in Physics*, 8:325–342, May–June 1994.

[27] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[28] William Hibbard. VisAD: Connecting people to computations and people to people. *ACM SIGGRAPH Computer Graphics*, 32(3):10–12, 1998.

[29] William L. Hibbard, Charles R. Dyer, and Brian E. Paul. A lattice model for data display. In R. Daniel Bergeron and Arie E. Kaufman, editors, *Proceedings of the IEEE Conference on Visualization 1994 (Vis '94)*, pages 310–317. IEEE Computer Society Technical Commitee on Computer Graphics/ACM SIGGRAPH, October 17–21 1994.

[30] William L. Hibbard, Charles R. Dyer, and Brian E. Paul. The VIS-AD data model: Integrating metadata and polymorphic display with a scientific programming language. In John P. Lee and Georges G. Grinstein, editors, *Proceedings of the IEEE Visualization 1993 Workshop on Database Issues in Data Visualization*, pages 39–68. Springer-Verlag, 1994.

[31] David M. Hilbert and David F. Redmiles. Extracting usability information from user interface events. *ACM Computing Surveys*, 32(4):384–421, December 2000.

[32] 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.

[33] T. J. Jankun-Kelly and Kwan-Liu Ma. Focus+Context display of the visualization exploration process. Technical Report CSE-2002-13, Computer Science Department, University of California, Davis, 2002.

[34] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, 1(3):26–49, August/September 1988.

[35] C. Charles Law, Amy Henderson, and James Ahrens. An application architecture for large data visualization: A case study. In *Proceesings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics (PVG '01)*, pages 125–159. IEEE Computer Society Technical Committee on Visualization and Graphics/ACM SIGGRAPH, 2001.

[36] John P. Lee and George G. Grinstein. An architecture for retaining and analyzing visual explorations of databases. In Gregory M. Nielson and Deborah Silver, editors, *Proceedings of the IEEE Conference on Visualization 1995 (Vis '95)*, pages 101–108. IEEE Computer Society, October 29–November 3 1995.

[37] John Peter Lee. *A Systems and Process Model for Data Exploration*. PhD thesis, U. of Massachuesetts Lowell, 1998.

[38] Marc Levoy. Spreadsheets for images. In Andrew Glassner, editor, *Proceedings of ACM SIGGRAPH 1994*, Computer Graphics Procceedings, Annual Conference Series, pages 139–146. ACM Press, 1994.

[39] Kwan-Liu Ma. Image Graphs—a novel approach to visual data exploration. In David Ebert, Markus Gross, and Bernd Hamann, editors, *Proceedings of the IEEE Conference on Visualization 1999 (Vis '99)*, pages 81–513. IEEE Computer Socieity Techincal Committee on Visualization and Graphics, October 24–29 1999.

[40] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, 1986.

[41] 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 ACM SIGGRAPH 1997*, Computer Graphics Procceedings, Annual Conference Series, pages 389–400. ACM SIGGRAPH, ACM Press/Addison-Wesley Publishing Co., 1997.

[42] Bruce H. McCormick, Thomas A. DeFani, and Maxine D. Brown. Visualization in scientific computing. *Computer Graphics*, 21(6), 1987.

[43] Patrick J. Moran and Chris Henze. Large field visualization with demand-driven calculation. In David Ebert, Markus Gross, and Bernd Hamann, editors, *Proceedings of the IEEE Conference Visualization 1999 (Vis '99)*, pages 27–34, New York, October 1999. ACM Press.

[44] Richard Olson. *Science Deified and Science Defied*, volume 1. University of California Press, 1982.

[45] Kannappan Palaniappan, A. Frederick Hasler, J. Fraser, and Michael 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*. American Meteorological Society, 2001.

[46] Penny Rheingans. Are we there yet? Exploring with dynamic visualization. *IEEE Computer Graphics and Applications*, 22(1):6–10, 2002.

[47] Steven F. Roth and Joe Mattis. Data characterization for intelligent graphics presentation. In *Conference Proceedings on Human Factors in Computing Systems (CHI'90)*, pages 193–200. ACM SIGCHI, ACM Press, April 1–5 1990.

[48] Daniel M. Russell, Mark J. Stefik, Peter Pirolli, and Stuart K. Card. The cost structure of sensemaking. In *Conference proceedings on Human factors in computing systems (CHI '93)*, pages 269–276. ACM Press, 1993.

[49] Greg Schussman, Kwan-Liu Ma, Davis Schissel, and Todd Evans. Visualizing DIII-D Tokamak magnetic field lines. In Bernd Hamann, Amitabh Varshney, and Thomas Ertl, editors, *Proceedings of the IEEE Conference on Visualization 2000 (Vis '00)*, Salt Lake City, October 2000. IEEE.

[50] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 12th IEEE Symposium on Visual Languages (VL '96)*, pages 336–343. IEEE Computer Society Technical Committee on Multimedia Computing, IEEE Computer Society Press, September 3–6 1996.

[51] Robert Spence. *Information Visualization*, page 92. ACM Press, 2001.

[52] Rebecca R. Springmeyer, Meera M. Blattner, and Nelson L. Max. A characterization of the scientific data analysis process. In Arie E. Kaufman and Gregory M. Neilson, editors, *Proceedings of the IEEE Conference on Visualization,1992 (Vis '92)*, pages 235–242. IEEE Computer Society Technical Committee on Computer Graphics, IEEE Computer Society Press, 1992.

[53] Ikuko Takanashi, Eric Lum, Kwan-Liu Ma, Joerg Meyer, Bernd Hamann, and Arthur J. Olson. Segmentation and 3d visualization of high-resolution human brain cryosections. In Robert F. Erbacher, Philip C. Chen, Matti Groehn, Jonathan C. Roberts, and Craig M. Wittenbrink, editors, *Visualization and Data Analysis 2002*, volume 4665, pages 55–61, Bellingham, Washington, 2002. SPIE- The International Society for Optical Engineering, SPIE.

[54] Soon Tee Teoh, Kwan-Liu Ma, Felix Wu, and X. Zhao. Case study: Interactive visualization for internet security. In Robert Moorhead, Markus Gross, and Kenneth I. Joy, editors, *Proceedings of IEEE Conference on Visualization 2002 (Vis '02)*, 2002.

[55] Lloyd Treinish. A function-based data model for visualization. In *Proceedings of IEEE Visualization '99 Late Breaking Hot Topics*, pages 73–76, 1999. Available at http://www.research.ibm.com/people/l/lloydt/dm/function/dm_fn.htm.

[56] 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. *Computer Graphics and Applications, IEEE*, 9(4):30–42, 1989.

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

[58] Stephen Wehrend and Clayton Lewis. A problem-oriented classification of visualization techniques. In Arie Kaufman, editor, *Proceedings of the First IEEE Conference on Visualization (Vis '90)*, pages 139–143, 469. IEEE Computer Society Technical Committee on Computer Graphics, IEEE Computer Society Press, October 23–26 1990.

[59] 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.

[60] Sherry Yang, Margaret M. Burnett, Elyon DeKoven, and Moshé Zloff. Representation design benchmarks: A design-time aid for VPL navigable static representation. *Journal of Visual Languages and Computing*, 8(5/6):563–599, October–December 1997.

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