VISUALIZING SCIENTIFIC COMPUTATIONS: A SYSTEM BASED ON LATTICE-STRUCTURED DATA AND DISPLAY MODELS

by

WILLIAM LOUIS HIBBARD

A dissertation submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN-MADISON

1995

VISUALIZING SCIENTIFIC COMPUTATIONS: A SYSTEM BASED ON LATTICE-STRUCTURED DATA AND DISPLAY MODELS

William Louis Hibbard

Under the supervision of Professor Charles R. Dyer

At the University of Wisconsin-Madison

Abstract

In this thesis we develop a system that makes scientific computations visible and enables physical scientists to perform visual experiments with their computations. Our approach is unique in the way it integrates visualization with a scientific programming language. Data objects of any user-defined data type can be displayed, and can be displayed in any way that satisfies broad analytic conditions, without requiring graphics expertise from the user. Furthermore, the system is highly interactive.

In order to achieve generality in our architecture, we first analyze the nature of scientific data and displays, and the visualization mappings between them. Scientific data and displays are usually approximations to mathematical objects (i.e., variables, vectors and functions) and this provides a natural way to define a mathematical lattice structure on data models and display models. Lattice-structured models provide a basis for integrating certain forms of scientific metadata into the computational and display semantics of data, and also provide a rigorous interpretation of certain expressiveness conditions on the visualization mapping from data to displays. Visualization mappings satisfying these expressiveness conditions are lattice isomorphisms. Applied to the data types of a scientific programming language, this implies that visualization mappings from data

aggregates to display aggregates can always be decomposed into mappings of data primitives to display primitives.

These results provide very flexible data and display models, and provide the basis for flexible and easy-to-use visualization of data objects occurring in scientific computations.

Charles R. Dyer, Professor, Computer Sciences, University of Wisconsin - Madison

Abstract

In this thesis we develop a system that makes scientific computations visible and enables physical scientists to perform visual experiments with their computations. Our approach is unique in the way it integrates visualization with a scientific programming language. Data objects of any user-defined data type can be displayed, and can be displayed in any way that satisfies broad analytic conditions, without requiring graphics expertise from the user. Furthermore, the system is highly interactive.

In order to achieve generality in our architecture, we first analyze the nature of scientific data and displays, and the visualization mappings between them. Scientific data and displays are usually approximations to mathematical objects (i.e., variables, vectors and functions) and this provides a natural way to define a mathematical lattice structure on data models and display models. Lattice-structured models provide a basis for integrating certain forms of scientific metadata into the computational and display semantics of data, and also provide a rigorous interpretation of certain expressiveness conditions on the visualization mapping from data to displays. Visualization mappings satisfying these expressiveness conditions are lattice isomorphisms. Applied to the data types of a scientific programming language, this implies that visualization mappings from data aggregates to display aggregates can always be decomposed into mappings of data primitives to display primitives.

These results provide very flexible data and display models, and provide the basis for flexible and easy-to-use visualization of data objects occurring in scientific computations.

Acknowledgments

I am greatly indebted to my advisor, Chuck Dyer, for showing me a different way of thinking about Computer Science problems, and for his consistent good nature.

Special thanks are due to Amir Assadi, Miron Livny, Tom Reps and Greg Tripoli for taking the time to review this work as members of my thesis committee, and to Tom DeFanti for his input as a guest committee member.

I want to thank Larry Landweber for suggesting that I pursue a doctorate and for introducing me to Chuck Dyer, long after I thought graduate school was behind me forever. I also want to thank Francis Bretherton, Bob Fox and John Anderson, the directors of the Space Science and Engineering Center where I am employed, for their support and encouragement. I especially want to thank Verner Suomi, the founder of the Space Science and Engineering Center, for his profound positive influence on my life over a period of many years.

Special thanks are due to Brian Paul, my principal collaborator in system development, and to Andre Battaiola, Dave Santek and Marie-Francoise Voidrot-Martinez for their collaboration in systems development.

I am indebted to Bob Rabin, Roland Stull, Bob Aune, Wilt Sanders, Dick Edgar, Mike Botts, Chris Crosiar, and all the other users of our systems for their helpful suggestions for improving our systems.

Marriage to AJ is the best thing that has happened in my life. The real work of this thesis was done at our home, and thus was always pleasant. Thanks to my sweet mother for always encouraging education and to my father for teaching me to figure things out for myself. This thesis is dedicated to Laura and Tommy - thanks to Jeannie and John for bringing them into the world.

Contents

Abstract ii

Acknowledgments iii

1. Introduction 1

1.1 Goals for Scientific Visualization 3

1.2 State of the Art in Scientific Visualization 10

1.2.1 The Data Flow and Object-Oriented Approaches 10

1.2.2 Data Models 13

1.2.3 Display Models 16

1.2.4 Automating the Design of Data Displays 18

1.3 Major Contributions 22

1.4 Thesis Outline 24

2. System Design for Visualizing Scientific Computations 25

2.1 A Scientific Computing Environment 25

2.2 Scientific Data 31

2.3 Scientific Displays 35

2.4 Mapping Data to Displays 37

3. An Analysis of Mappings from Data to Displays 40

3.1 An Analytic Approach Based on Lattices 41

3.1.1 Basic Definitions for Ordered Sets 42
3.1.2 Scientific Data Objects as Approximations of Mathematical Objects 44
3.1.3 A Mathematical Structure Based on the Precision of Scientific data 46
3.1.4 Data Display as a Mapping Between Lattices 56
3.2 A Scientific Data Model 59
3.2.1 Interpreting the Data Model as a Lattice 64
3.2.2 Defining the Lattice Structure 66
3.2.3 Embedding Scientific Data Types in the Data Lattice 70
3.2.4 A Finite Representation of Data Objects 74
3.3 A Scientific Display Model 75
3.4 Scalar Mapping Functions 79
3.4.1 Structure of Display Functions 79
3.4.2 Behavior of Display Functions on Continuous Scalars 82
3.4.3 Characterizing Display Functions 85
3.4.4 Properties of Scalar Mapping Functions 87
3.5 Principles for Scientific Visualization 91

4. Applying the Lattice Model to the Design of Visualization Systems 94
4.1 Integrating Metadata with a Scientific Data Model 96
4.2 Interacting with Scientific Displays 105
4.3 Visualizing Scientific Computations 123
4.4 System Organization 144

5. Applying the Lattice Model to Recursive Data Type Definitions **148**

5.1 Recursive Data Type Definitions 148

5.2 The Inverse Limit Construction 149

5.3 Universal Domains 151

5.4 Display of Recursively Defined Data Types 153

6. Conclusions **155**

6.1 Main Contributions and Limitations 155

6.2 Future Directions 158

A. Definitions for Ordered Sets **161**

B. Proofs for Section 3.1.4 **165**

C. Proofs for Section 3.2.2 **170**

D. Proofs for Section 3.2.3 **178**

E. Proofs for Section 3.2.4 **186**

F. Proofs for Section 3.4.1 **190**

G. Proofs for Section 3.4.2 **204**

H. Proofs for Section 3.4.3 **215**

I. Proofs for Section 3.4.4 **223**

Bibliography **232**

List of Figures

1.1. Image data displayed in four different ways. 6

1.2. Interactive display of the output of a numerical weather model. 8

1.3. The place of visualization in the computational process. 10

1.4. A simple rendering pipeline for three-dimensional graphics. 11

1.5. Bertin's display model. 17

2.1 The place of visualization in the computational process. 27

2.2. A snapshot of an executing shallow fluid simulation model. 29

3.1. Order relation of a continuous scalar. 48

3.2. Approximating real functions by arrays. 49

3.3. Order relation of arrays. 50

3.4. Least precise image in sequence of four. 51

3.5. Second image in sequence of four, ordered by precision. 52

3.6. Third image in sequence of four, ordered by precision. 53

3.7. Most precise image in sequence of four. 54

3.8. Meaning of the expressiveness conditions. 58

3.9. Order relation of a discrete scalar. 60

3.10. Order relation of tuples. 62

3.11. Embedding a tuple type into a lattice of sets of tuples. 65

3.12. Embedding an array type into a lattice of sets of tuples. 66

3.13. Defining an order relation on sets of tuples. 70

3.14. The roles of display scalars in a display model. 77

3.15. Mappings from scalars to display scalars. 82

3.16. The behavior of a display function on a continuous scalar. 85
4.1. An image displayed in a Cartesian coordinate system. 102
4.2. An image displayed in a spherical Earth coordinate system. 103
4.3. Three-dimensional radar data. 104
4.4. X-ray events from interstellar gas. 106
4.5. A *goes_sequence* object displayed as a terrain. 111
4.6. A *goes_sequence* object displayed in four different ways. 115
4.7. Three views of chaos. 120
4.8. Visualizing the computations of a bubble sort algorithm. 126
4.9. Visually experimenting with algorithms. 127
4.10. Visually tracing back to the causes of computational errors. 128
4.11. A close-up view of two regions of a *goes_partition* object. 132
4.12. A close-up view restricted to "cloudy" pixels. 133
4.13. Three *ir_image_partition* objects displayed as terrains. 134
4.14. Three *histogram* objects displayed as graphs. 135
4.15. A two-dimensional histogram of X-ray events. 140
4.16. Visualizing the three criteria used to select cumulus clouds. 143
4.17. VisAD system organization. 147
5.1. A type construction operator represented by a function. 151

Chapter 1

Introduction

Physical scientists observe nature, formulate laws to fit the observations, and predict future observations in order to test their laws. Mathematics is the language of observations, laws and predictions, but the complexity of modern science demands that mathematical calculations be automated using computers. The number of observations of nature dictates that they are analyzed by computer algorithms, and the number of computations required to predict nature dictates that predictions are made by numerical simulation models running on computers. Thus computers have become essential tools for scientists for both observing and simulating nature.

In spite of their essential role, computers are also barriers to scientific understanding. Unlike hand calculations, automated computations are invisible, and, because of the enormous numbers of individual operations in automated computations, the relation between an algorithm's input and output is often not intuitive. This problem was discussed in a report to the National Science Foundation (McCormack, DeFanti and Brown, 1987) and is illustrated by the behavior of meteorologists responsible for forecasting weather. Even in this age of computers, many meteorologists manually plot weather observations on maps and then draw iso-level curves of temperature, pressure and other fields by hand (special pads of maps are printed for just this purpose). Similarly, radiologists use computers to collect medical data, but are notoriously reluctant to apply image processing algorithms to those data. To these scientists with life and death responsibilities, computer algorithms are black boxes that increase rather than reduce risk.

The barrier between scientists and their computations is being bridged by *scientific visualization* techniques that make computations visible. Scientific visualization is itself a computational process that transforms the data objects of scientific computations into visible images on a computer display screen. Scientific visualization is difficult because of the variety and complexity of scientific data, because the variety of scientific problems implies that scientists need to see the same data in many different ways, and because scientists need tools that are easy to use so that they can concentrate on understanding their computations rather than understanding their visualization tools.

The size of scientific data sets is often used to justify the development of scientific visualization, and it is true that scientists need to be able to see large data sets. However, the more important motive for visualization is the invisibility of automated computations. To see this, consider the volumes of satellite images of the Earth. A pair of GOES (Geostationary Operation Environmental Satellite) located at East and West stations over the U.S. generate one 1024 by 1024 image every four seconds. NASA's Earth Observing System, as planned, will generate about five 1024 by 1024 images per second. These data volumes are so large that they will overwhelm any scientist trying to look at them all. Furthermore, these images are quantitative measurements rather than just pictures. The real value of these images must be extracted by automated computations that can process the images faster than a person can coherently look at them. Thus the work of Earth scientists is to develop algorithms for this automated processing, and the proper role of visualization is helping scientists to understand how their algorithms work and how to improve them.

1.1 Goals for Scientific Visualization

Scientific data exist in a wide variety of structures. A few examples include two-dimensional images:

```
type image = array [row] of array [column] of radiance;
```

three-dimensional grids:

```
type grid = array [row] of array [column] of array [level] of temperature;
```

time sequences of images and grids:

```
type image_sequence = array [time] of image;
type grid_sequence = array [time] of grid;
```

images and grids with multiple values per pixel:

```
type multi_image = array [row] of array [column] of
        structure {ir_radiance; vis_radiance};
type multi_grid = array [row] of array [column] of array [level] of
        structure {pressure; temperature; humidity};
```

irregularly located data such as observations made by ships or aircraft:

```
type observations = array [index] of structure {latitude; longitude; altitude; pressure};
```

one-dimensional and multi-dimensional histograms derived from other data:

```
type histogram_1d = array [temperature] of count;
type histogram_2d = array [temperature] of array [pressure] of count;
```

and partitions of images and grids into spatial regions:

```
type image_partition = array [region] of image;
type grid_partition = array [region] of grid;
```

Furthermore, physical systems are observed by collections of instruments so the observed state of a physical system is a complex combination of data sensed by different types of instruments. Similarly, simulations generate complex combinations of data describing interacting physical systems (e.g., atmospheric physics and chemistry, ocean physics and chemistry, and land and ocean surface processes). Scientific data are made more complex because of scientists' need to precisely document where, when and how they were obtained (this documentation is a form of *metadata*, and must be considered as part of scientific data). The first goal of this thesis is to develop visualization techniques that

1. Can be applied to the data of a wide variety of scientific applications.

Scientists need to see the same data displayed in different ways, depending on what kinds of information they are looking at. For example, Figure 1.1 shows a time sequence of multi-variate image data displayed in four different ways. The upper-left

window shows radiance values as colors, which is appropriate for seeing spatial patterns and textures. The upper-right window shows infrared radiances as a terrain (colored by visible radiances), appropriate for seeing slopes. The time sequence can be animated in the upper-right and upper-left windows, which is appropriate for seeing motion. Alternatively, the time sequence is stacked up along the vertical axis in the lower-right window, which is appropriate for looking closely at rates of motion and changes in shape and intensity. Information about the spatial locations of pixels is not shown in the lower-left window, producing a colored three-dimensional scatter diagram which is appropriate for seeing correlations among infrared radiance, visible radiance, variance and texture (variance and texture are derived from infrared radiance). Each of the four views presented in Figure 1.1 is appropriate for seeing a different aspect of the same data. More generally, the primary reason scientists use scientific visualization is to find unexpected patterns in data, since expected patterns can just be measured and characterized by statistical calculations applied to data. And flexibility in the ways that data are displayed is critical in the search for unexpected patterns. Thus the second goal of this thesis is to develop visualization techniques that

2. Can produce a wide variety of different visualizations of data appropriate for different needs.

Figure 1.1. A time sequence of multi-variate image data displayed in four different ways. (color original)

Because of the large volumes of scientific data it is often impossible to display a data object in a single image or even a single animation sequence. Instead, scientists need to interactively explore large data objects. For example, Figure 1.2 shows a snapshot of an interactive animated display of the output of a numerical weather model. The white object is a balloon seven kilometers high in the shape of a squat chimney that floats in the air above a patch of tropical ocean. The purpose of the numerical simulation is to verify that, once air starts rising in the chimney, the motion will be self-sustaining and create a perpetual rainstorm. The vertical color slice shows the distribution of heat (and when animated shows the flow of heat), the yellow streamers show the corresponding flow of air up through the chimney, and the blue iso-surface shows the precipitated cloud ice (a cloud water iso-surface would obscure the view down the chimney, so it is not shown in this snapshot). Viewers of this visualization can interactively move the color slice in the three-dimensional box of atmosphere, can interactively release new streamers in the air flow, can interactively change the value of the cloud ice iso-surface, and can rotate and zoom the box in three dimensions. They can choose different combinations of fields to display, and can choose the ways that each field is depicted (e.g., color slice, iso-surface, contour slice). Such interactivity is critical for allowing scientists to search through large amounts of data for unexpected patterns. Hence, the third goal of this thesis is to develop visualization techniques that

3. Enable users to interactively alter the ways data are viewed.

Figure 1.2. A snapshot of an interactive animated display of the output of a numerical weather model. (color original)

Because visualization is used for communicating results of observations and computations to scientists, they need to be able to control it themselves (that is, they cannot delegate expertise with visualization to support staff). In order not to distract scientists from the difficult task of understanding data, visualization must be easy to control. Thus the fourth goal of this thesis is to develop visualization techniques that

4. Require minimal effort by scientists.

As stated at the start of this section, the rationale for visualization is scientists' need to see the results of computations. Thus visualization is intimately connected with computation. Just as the complexity of data requires that the visualization process should be interactive, the complexity of computation requires that the overall computational process, which includes visualization, should be interactive. Figure 1.3 illustrates the interactive cycle of the computation process. If these three activities are done in separate software systems, then scientists must repeatedly switch between systems and manage the movement of information between these systems. This user overhead can be reduced by integrating all three activities in one system. Furthermore, visualization can be especially useful during program execution, allowing users to dynamically monitor intermediate results of computations and respond by immediately adjusting parameters of those computations. This is sometimes called *computational steering*. The fifth goal of this thesis is to develop visualization techniques that

5. Can be integrated with a scientific programming environment.

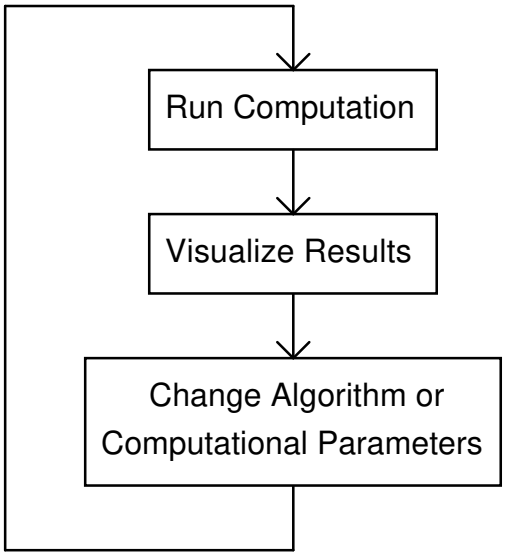

Figure 1.3. The place of visualization in the computational process.

1.2 State of the Art in Scientific Visualization

Here we consider the state of the art in scientific visualization and how well current techniques achieve our goals.

1.2.1 The Data Flow and Object-Oriented Approaches

Visualization research has focused primarily on developing specialized visualization techniques suited to specific types of data. However, some research has sought common patterns in the ways that displays are computed. For example, the *rendering pipeline* is a widely applicable abstraction for the ways that data are transformed into displays. Figure 1.4 illustrates a simple rendering pipeline:

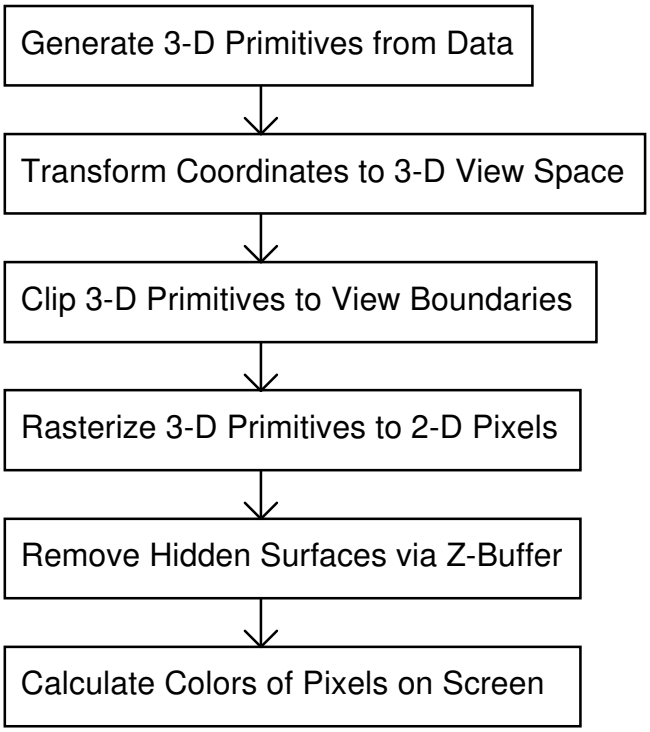

Figure 1.4. A simple rendering pipeline for three-dimensional graphics.

The FRAMES system abstracted the rendering pipeline to let users specify display processes as sequences of UNIX filters (Potmesil and Hoffert, 1987). The GRAPE system introduced branching into these data transformations and let users define display processes as acyclic graphs of modules (Nadas and Fournier, 1987). The ConMan system provided a graphical user interface for specifying display processes as networks of modules (Haeberli, 1988). This idea has been adopted as the basis of several widely-used data flow visualization systems, such as AVS (Upson et al., 1989) and Khoros (Rasure et al., 1990). These data flow systems provide large libraries of modules that implement basic computational and display operations, and also provide graphical user interfaces for synthesizing complex visualization algorithms from these module libraries.

The recognition that different kinds of displays are generated by similar sets of operations also led to the object-oriented approach to synthesizing visualization mappings. The object-oriented approach uses inheritance and polymorphism to exploit the common properties and natural hierarchy of data displays. The Powervision system, for example, used an object-oriented language to support interactive development of image processing algorithms (McConnell and Lawton, 1988). The system defined a set of primitive generic functions for accessing data objects (for example, for iterating over parts of objects, for checking boundary conditions, etc.). Algorithms for synthesizing displays were expressed in terms of these generic functions. As users defined new object classes they could apply existing display algorithms to those classes as long as the new classes included definitions for the generic functions for accessing data objects.

The SuperGlue system was developed as a programming environment for developing scientific visualization applications based on Scheme, C and the GNU Emacs editor (Hultquist and Raible, 1992). It defined a class hierarchy for various types of scientific data objects and displays. User extensions to this class hierarchy could take advantage of inheritance and polymorphism to simplify their definition.

The VISAGE system implemented a hierarchy of over 500 classes for both process objects and data objects (Schroeder, Lorenson, Mantanaro and Volpe, 1992). The process objects implemented the visualization process as data flow networks of simpler processes. The data objects implemented a variety of scientific data organizations and a variety of display organizations.

While these systems have been useful to scientists, their approach to generality is through the enumeration of data types and the enumeration of display techniques. Thus these systems have become very large and complex. Furthermore, scientists must spend considerable effort to produce visualizations using these systems. While scientists could

explore different ways of displaying data by interactively changing the data flow networks that transform data into displays, in practice they do not. Rather, support staff design data flow networks and scientists use them to generate fixed types of displays from data. Similarly for the object-oriented systems. The developers of the VISAGE system described one visualization application of their system that required 12,000 lines of code specific to the application. Scientists need visualization techniques that let them change the way that they look at data without understanding complex programs or data flow networks.

1.2.2 Data Models

Rather than approaching generality by enumerating data types and display techniques, we can achieve generality through the abstraction of data and displays. That is, by developing broadly applicable abstract models of scientific data and displays, we can systematically study the ways that visualization processes transform data into displays. A *data model* defines a set of data objects, the way data objects are organized in the set, and operations on the data objects (often by reference to their internal structures). Data models have been the subject of several recent workshops and publications (Treinish, 1991; Haber, 1991; Robertson et al., 1994, Lee and Grinstein, 1994). *Display models* are similar to data models and are discussed in the next section.

The requirements for a scientific data model can be understood in terms of the role of data in science. Scientists design mathematical models of nature. These models identify numerical variables (e.g., *time*, *altitude*, *temperature*) and functional relations between these variables (e.g., *temperature* as a function of *time*). These models define the states of nature as vectors of variables and functions. For example, the state of a point in the atmosphere is a vector of variables such as *temperature*, *pressure*, *humidity* and *wind*

velocity, and the state of the entire atmosphere is a vector of functions. We use the term *mathematical objects* to denote the numbers, functions and vectors of mathematical models. When scientists want to use their mathematical models to analyze a set of observations, or to simulate a physical system, they implement their models as computer programs. Mathematical objects are represented by *scientific data objects* in these implementations, and therefore a *scientific data model* should reflect the ways that scientific data objects represent mathematical objects. There are many ways to define scientific data models. However, any scientific data model will incorporate the following components:

1. The types of *primitive values* occurring in data objects. These represent primitive variables defined in mathematical models of nature. Thus a data model may define a *floating-point* type to represent real variables such as *time* and *temperature*, may define an *integer* type to represent integer variables such as an *event_count*, and may define a *string* type to represent names such as city or state names. A rigorous data model specifies the relations and operations defined on values of primitive types. The definition of a type of primitive value may include arithmetical operations, string operations, an order relation, a metric or a topology.

2. The ways that primitive values are *aggregated* into data objects. These aggregates represent complex mathematical objects, such as vectors, functions, vectors of functions, and so on. There are a variety of approaches to defining data aggregates. In the C programming language, vectors can be represented by *structures*, functions can be represented by *arrays*, and *pointers* can be used to define complex networks of values. Most programming languages provide a few simple data structuring rules

that can be combined to define a wide variety of data aggregates. On the other hand, most scientific analysis and visualization systems support specific types of aggregates based on particular application needs. These may include two-dimensional images (as generated by satellites and other observing systems), three-dimensional grids (as generated by numerical simulations and some observing systems), and vector and polygon lists (generated by applying visualization operators to images and grids, and by map makers).

3. *Metadata* about the relation between data and the physical things they represent. For example, given a meteorological temperature value, metadata includes the fact that it is a temperature, its scale (Fahrenheit, Kelvin, etc.), its spatial and temporal location in the Earth's atmosphere, and whether it is a point sample or an average over space and time. Temperature values have limited accuracy, whether sensed by an instrument or computed by a weather model, and an estimate of accuracy is another form of metadata. Because instruments and observing systems are fallible, an expected data value may not be defined at all, so *missing* data indicators are a form of metadata. If a temperature is observed by an instrument, there may be metadata about the instrument (for example, aperture, pointing direction, filters, etc.). If a temperature is computed from other values, there may be metadata about the algorithm used to compute it and the source of the algorithm's inputs.

The term *metadata* has several different meanings. It is sometimes denotes information about the organization of data, in which case it may be called *syntactic metadata*. Here it denotes information about the meaning of data, and may be called *semantic metadata*. We can think of metadata as secondary data that are critical to the

usefulness of primary data. For example, while a satellite image may primarily consist of an array of pixel radiance values, those data are scientifically useless without other arrays that specify the Earth locations of pixels, how pixel values correspond to physical radiances, and so on.

1.2.3 Display Models

Just as we can define systematic models of scientific data, we can define systematic models of scientific displays. In particular, it is useful to note that computer programs generate displays in the form of data objects. Bertin's detailed display model, first published in 1967, illustrates how a display model addresses the issues of primitives and aggregates (Bertin, 1983). Bertin defined a display as an aggregate of *graphical marks*, and identified eight *primitive variables* of a graphical mark: two spatial coordinates of the mark in a graphical plane (he restricted his attention to static two-dimensional graphics), plus the mark's size, value, texture, color, orientation, and shape. Bertin defined diagrams, networks and maps as spatial aggregates of graphical marks. Figure 1.5 illustrates Bertin's display model.

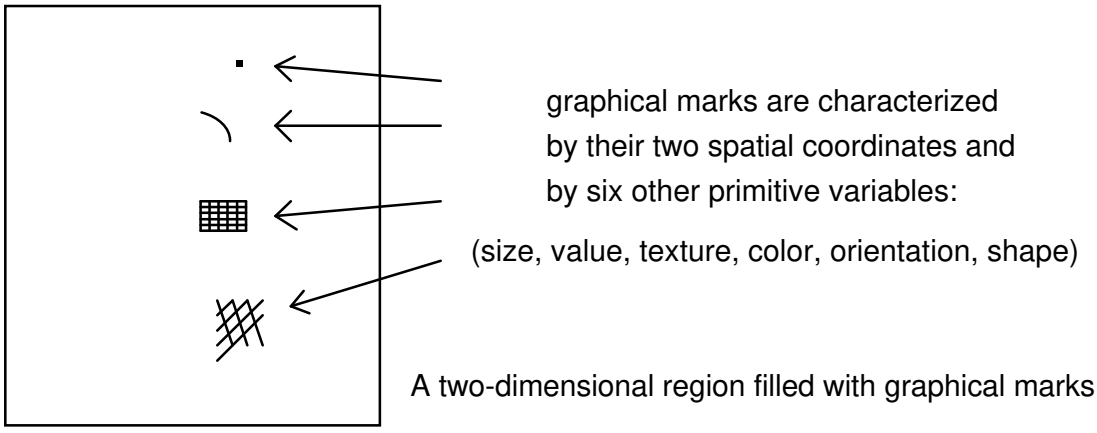

Figure 1.5. Bertin's display model. He modeled displays as sets of graphical marks in a two-dimensional spatial region.

Bertin's display model was limited to static two-dimensional displays. This corresponds to what can be physically displayed on a two-dimensional screen at one time. However, computer-generated displays generate the illusion of three dimensions and show motion by changing screen contents at short intervals. We can even regard various forms of user interaction as an integral part of the display. Thus we distinguish between physical and logical display models. We let V' denote the set of physical displays, which are two-dimensional and static, and we let V denote the set of logical displays, which are three-dimensional, animated and interactive. The mapping $RENDER : V \rightarrow V'$ includes traditional graphics operations such as iso-surface generation, volume rendering, projection from three to two dimensions (rotate, zoom and translate), clipping, hidden-surface removal, shading, compositing, and animation (these operations could be implemented in a rendering pipeline, as illustrated in Figure 1.4). A changing set of mappings, $RENDER : V \rightarrow V'$, expresses the three-dimensional, animated, interactive nature of logical displays in V.

Visualization is a process that maps data objects to displays. We let U' denote a set of mathematical objects, and we let U denote a set of data objects used to represent them. Then the overall visualization process may be viewed as a sequence of mappings $U' \rightarrow U \rightarrow V \rightarrow V'$. The mapping from U' to U expresses the way that scientists implement their mathematics on computers, and the mapping from V to V' is the generally well understood physical display generation process (Foley and Van Dam, 1982; Lorensen and Cline, 1987). Thus we will concentrate our interest on the mapping $D : U \rightarrow V$. In order to optimize the generality of visualization techniques to different scientific applications, we seek scientific data models U whose primitive values are defined in terms of abstract mathematical properties, whose aggregates are constructed using a few simple rules that can be combined in complex ways, and that integrate a variety of metadata. We also seek display models V that are abstract and that include interactive displays.

While the proper abstractions for U and V are necessary for display techniques that are flexible and easy to use, the proper abstraction for the mapping D is also necessary. In the next section we describe efforts to automate the choice of this mapping.

1.2.4 Automating the Design of Data Displays

As described in Section 1.2.1, the object-oriented and data flow approaches define natural methodologies for designing programs (or data flow diagrams) for transforming data into displays, but they still require considerable programming effort from their users. In response to scientists' need for visualization techniques that are easy to use, there have been a variety of efforts to automate the design of algorithms for producing data displays. This goal is often called *automating the design of data displays*, since the research focuses on automating the choice among the many different ways of displaying the same data.

Mackinlay sought to automate the design of displays for data from relational database systems (Mackinlay, 1986). His technique combined a relational data model with Bertin's display model. A relation is a set of tuples of values. Sets of primitive values called *domains* are defined for each position in a relation's tuples. Mackinlay classified domains as nominal (without an order relation), ordinal (with an order relation but without a metric or arithmetical operations) or quantitative (with a metric and arithmetical operations). These primitive values are aggregated into sets of tuples to form relations. Mackinlay's data model also allowed functional dependencies to be defined between the domains of a relation (these are restrictions on the sets of tuples that may form relations).

Mackinlay modeled displays as sentences in a graphical language. Sentences were sets of 2-tuples, where each tuple pairs a graphical mark with a two-dimensional screen location. He also attached attributes to graphical marks for specifying their size, color, orientation, etc. The values of these attributes are similar to the primitive values Bertin used for graphical marks. Thus, in Mackinlay's model a display could be interpreted as a set of tuples, where each tuple contains two screen coordinates and the values of the various attributes of a graphical mark.

Mackinlay defined expressiveness and effectiveness criteria for the mapping from data relations to display sentences. The expressiveness criteria require that a display sentence:

1. Encodes all the facts in a set (that is, the set of facts about a data relation), and

2. Encodes only the facts in a set.

The effectiveness criteria provide a way to choose between different display sentences that satisfy the expressiveness criteria. For example, an effectiveness criterion may specify that quantitative information is easier to perceive when encoded as spatial position rather than as color. Mackinlay also solicited visualization goals from the user. Effectiveness criteria and visualization goals were expressed formally in terms of predicates and functions applied to relational data and display sentences. These were used as the basis for a backtracking search for an optimal display.

Mackinlay's display model was static and two-dimensional and therefore too limiting for scientific visualization. Furthermore, while the relational model can, in theory, be used for scientific data, it does not naturally fit the ways that scientific data are aggregated. Robertson (Robertson, 1991), Senay and Ignatius (Senay and Ignatius, 1991; Senay and Ignatius, 1994), and Beshers and Feiner (Beshers and Feiner, 1992) all sought automated techniques for designing displays for scientific data.

Robertson's data model classified primitive values as either nominal or ordinal. Nominal values were further classified as single or multiple valued (that is, sets of values) and ordinal values were classified as discrete or continuous (this is a classification of the topology of primitive value sets). Primitive values were aggregated as distributions over an *n*-dimensional space. Robertson modeled displays as two-dimensional and three-dimensional surfaces and their attributes (for example, color and texture). His methodology solicited a set of visualization goals from the user, in terms of the scales of the user's interest (that is, point, local or global) in different variables, and in terms of the user's interest in correlations between various pairs of variables. Data displays were generated by matching data attributes and relations to display attributes and relations, according to the user's visualization goals.

Senay and Ignatius' data model classified primitive values as qualitative or quantitative, and aggregated primitive values as functional dependencies between variables. Their data model included metadata for coordinate systems and data sampling. Senay and Ignatius modeled displays using Bertin's graphical marks, and using specific aggregates of marks (for example, icons and iso-surfaces). These were further classified as to whether they encoded a single variable or multiple variables. Displays were generated by applying production rules for matching data characteristics with display characteristics.

Beshers and Feiner's data model consisted of functions from one set of real variables to another. Their display model sought to overcome the limitation to three spatial axes by embedding small spatial coordinate systems (that is, small sets of graphical axes) within larger spatial coordinate systems. Their display model formalized interactive exploration of data by allowing the user to move small coordinate systems around within larger coordinate systems. Their technique searched through a large set of possible designs, evaluating them based on a set of user-defined visualization tasks.

While all of these efforts sought to automate the design of displays, their display models were limited to specific types of displays and they enumerated specific display techniques as the search spaces for their automated techniques. That is, their focus was to automate the user's task of choosing among enumerated sets of visualization techniques. In the next section we describe an alternative approach that defines certain general analytic conditions on the mapping from data to displays, and then derives visualization mappings that satisfy those conditions.

In each of the previous automated approaches described above, displays were designed based on information about visualization goals provided by the user. Obviously, some form of user input is necessary for users to be able to control display design.

However, user interface issues are notoriously complex and it is not obvious that an encoding or parameterization of visualization goals is the most effective way for users to control visualization systems. It may be most effective to allow users to make their own translation from their goals to some other form of controls over visualization. In particular, an interactive system that lets users experiment with various ways of displaying their data may be more effective than an automated system. An interactive system enables users to experiment with small changes to their display controls and to see the effect of those changes on the way that their data are displayed. Such experimentation is also often the fastest way for scientists to learn how a visualization system works.

1.3 Major Contributions

The main contributions of this thesis can be summarized as follows:

1. Development of a system for scientific visualization that enables a wide variety of visual experiments with scientific computations. This system integrates visualization with a scientific programming language that can be used to express scientific computations. This programming language supports a wide variety of scientific data types and integrates common forms of scientific metadata into the computational and display semantics of data. Any data object defined in a program in this language can be visualized in a wide variety of ways during and after program execution. The controls for data display are simple and independent of data type. Displays are controlled by a set of simple mappings rather than program logic. These mappings are independent of data type and separate from a user's scientific programs, which is a clear distinction from previous visualization systems that require scientists to embed calls to visualization functions in their programs. Furthermore, computation

and visualization are highly interactive. In particular, the selection of data objects for display and the controls for how they are displayed are treated like any other interactive display control (e.g., interactive rotation). Previous visualization systems require a user to alter his program in order to make such changes. The generality, integration, interactivity and ease-of-use of this system all enhance the user's ability to perform visual experiments with their algorithms.

2. Introduction of a systematic approach to analyzing visualization based on lattices. We define a set U of data objects and a set V of displays and show how a lattice structure on U and V expresses a fundamental property of scientific data and displays (namely that they are approximations to the physical world). The visualization repertoire of a system can be defined as a set of mappings of the form $D : U \rightarrow V$. It is common to define a system's visualization repertoire by enumerating such a set of functions. However, an enumerated repertoire is justified only by the tastes and experience of the people who decide what functions to include in the set. In contrast, we interpret certain well-known expressiveness conditions on the visualization mapping $D : U \rightarrow V$ in terms of a lattice structure, and define a visualization repertoire as the set of functions that satisfy those conditions. Such a repertoire is justified by the generality of the expressiveness conditions. We show that visualization mappings satisfy these conditions if and only if they are lattice isomorphisms. Lattice structures can be defined for a wide variety of data and display models, so this result can be applied to analyze visualization repertoires in a wide variety of situations.

3. Demonstration of a specific lattice structure that unifies data objects of many different scientific types in a data model *U*, and demonstration that the same lattice structure can express interactive, animated, three-dimensional displays in a display model *V*. These models integrate certain kinds of scientific metadata into the computational and display semantics of data. In the context of these scientific data and display models, we show that the expressiveness conditions imply that mappings of data aggregates to display aggregates can always be factored into mappings of data primitives to display primitives. We show that our display mappings are complete, in the sense that we characterize all mappings satisfying the expressiveness conditions.

1.4 Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2 we describe the architecture of a system for scientific visualization based on the goals described in Section 1.1. As described in Section 1.2, current visualization systems approach the goals for flexibility by enumerating different data types and different types of displays. In Chapter 3 we develop an alternate approach to flexibility based on defining very general conditions on the mapping from data to displays, and we analyze the repertoire of functions that satisfy those conditions. We summarize the results of this analysis in terms of a set of principles for visualization. In Chapter 4 we continue the presentation of our visualization system architecture based on those principles. In Chapter 5 we discuss how the analysis of Chapter 3 might be extended to data and display models appropriate for general programming languages. Chapter 6 summarizes the conclusions of this thesis.