

# Expressiveness of the Data Flow and Data State Models in Visualization Systems

Ed H. Chi  
Palo Alto Research Center  
3333 Coyote Hill Road, Palo Alto, CA 94304  
echi@parc.com

## ABSTRACT

Visualization can be viewed as a process that transforms raw data (value) into views. There has been two major category of data process models that have been proposed to model the visualization transformation process. This paper seeks to compare the Data Flow Models and the Data State Models. Specifically, it proves that, in terms of expressiveness, anything that can be represented using the Data Flow Model can also be represented using the Data State Model, and vice versa.

**Categories:** I.3.6 Graphics Methodology and Techniques, H.5.2 User Interfaces, H.5.1 Multimedia Information Systems

**General Terms:** Design, Human Factors, Theory, Verification

## 1. INTRODUCTION

*Graphics is a very simple language. Its laws become self-evident when we recognize that the image is transformable, that it must be reordered, and that its transformations represent a visual form of information-processing.* —Jacques Bertin [2, p. 183]

Visualization transforms data or information into graphical forms to be represented on the computer display. Hence, visualization deals with both *transformations* and *representations*. Transformation is the process that converts data into graphical primitives. Representation is the data structures that are used to handle and store the various outputs of these processes.

Traditional visualization data flow networks [14, 16, 9, 11, 1, 10] have concentrated on the various transformations that are necessary to generate a computer display. These data flow networks are typically depicted graphically by drawing a network with nodes representing data transformation processes, and directional edges representing how data flows from one process to another. Experience in the visualization field has shown that the Data Flow Model is an effective visual programming model that lets users build an application by integrating modular components.

We have introduced the Data State Model in previous papers [6, 5, 4, 3]. At first glance it has very similar characteristics to the Data Flow Model. For instance, both models use nodes and edges to transcribe the visualization process. Both models use these graphs to describe how data sets travel through the processing mechanism.

However, there are some important differences. Data State Model captures distinct data states, whereas the Data Flow Model captures the order of distinct processes that comprise of a visualization. This difference is reflected in that the Data Flow Model uses nodes to denote processes and edges to denote data flow directions, whereas the Data State Model uses nodes to denote data states and edges to denote processes. Data State Models have the distinct advantage of representing *Data States* explicitly.

Given these differences, our research question is: “What is the relationship between the functionality of the Data State Model and the Data Flow Model?” Is one model more expressive than the other? If we are given a Data Flow Model, can we build a Data State Model that produces the same output? For example, are there certain visualization constructions that are not possible with the Data State Model? User experiences and commercial success have established the capability of data flow visualization systems and the expressiveness of the Data Flow Model. By comparing the Data State Model with it, we can learn the merits of each model. In this paper, we show that the Data State Model is as expressive as the Data Flow model, and vice versa. We present a duality transformation between the two models. Using the duality transformation, we show that the two models are equally expressive.

## 2. RELATED WORK

As discussed earlier, since visualization can be viewed as a transformation process that converts data values into graphical views, researchers have generally used a graph model to describe the transformations. The advantage of a graph model is that it is very easy to understand a node-and-link diagram, even for novice users, because of its familiarity. Here we will review these models.

The first graph model proposed for modeling visualization processes is the Data Flow Model [16, 9, 11], which has since been accepted as an industry standard way of constructing visualization for large scientific data sets. In scientific visualization, many researchers and practitioners have examined the use of a data flow network for constructing visualizations [16, 9, 11]. The representation used in this graph model is that: (a) nodes represented using boxes are process stages that can be applied to data, (b) edges represented using arrows are the direction in which data moves to go on to the next stage. Also, sometimes (c) the input and output ports on each node have data types associated with it, and the data types must match in order for two ports to be connected with each other.

The Data Flow Model is well-known in the visualization field, thus we will not go over the model in too much detail here. We would like to mention, however, a good open-source project implementing some of these ideas [13]. Schroeder et. al. described a conceptual data flow model in the context of a scientific visualization toolkit for applying operations to generate a visualization. The

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2001 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

model consists of a visualization network that can contain multiple sources and sinks. Every step in the middle of the network consists of filters that have inputs and outputs.

More recently, Data State Models [5] have also been proposed recently, most notably for information visualization systems [6]. Figure 1 shows an example of the Data State Model, which breaks down each technique into four Data Stages, three types of Data Transformation and four types of Within Stage operators. The visualization data pipeline is broken into four distinct Data Stages: Value, Analytical Abstraction, Visualization Abstraction, and View (See Table 1). Transforming data from one stage to another requires one of the three types of Data Transformation operators: Data Transformation, Visualization Transformation, and Visual Mapping Transformation (Table 2).

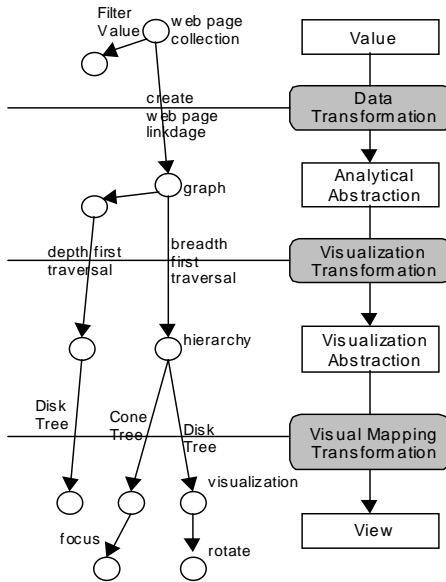


Figure 1: The Web Analysis Visualization Pipeline

Stage	Description
Value	The raw data.
Analytical Abstraction	Processed data that is not yet mappable but include all information from the raw data that will be visualized.
Visualization Abstraction	Information that is mappable and visualizable on the screen using at least one visualization technique.
View	The end-product of the visualization mapping, where the user sees and interprets the picture presented to her.

Table 1: Information Visualization Pipeline stages

Processing Steps	Description
Data Transformation	Generates some form of analytical abstraction from the value (usually by extraction).
Visualization Transformation	Takes an analytical abstraction and further reduce it into some form of visualization abstraction, which is visualizable content.
Visual Mapping Transformation	Takes information that is in a visualizable format and presents a graphical view to the user.

Table 2: Transformation Steps in InfoVis Pipeline

Within each Data Stage, there are also operators that do not change the underlying data structures. These are the Within Stage

Operators, of which there are four types, corresponding to the four Data Stages: Within Value, Within Analytical Abstraction, Within Visualization Abstraction, and Within View.

Figure 1 shows an example of the Data State Model applied to the problem of visualizing the connections between a set of Web pages. This example shows that: (1) some operators create new kinds of data sets, whereas some operators create filtered subsets, which is the difference between Transformation and Within Stage operators, and (2) that the same Visualization Abstractions can be mapped using a variety of Visual Mapping Transformation operators. For example, Disk Trees or Cone Trees can both be applied to a hierarchy of interconnected nodes.

Prior to this work, some researchers have hinted at the use of a Data State Model to represent visualization processes. For example, Lee and Grinstein [12] presented a conceptual model for database visual exploration, which describes the analysis process as a series of value-to-value, value-to-view, view-to-value, and view-to-view transformations. They also describe the concept of generating metadata using database queries to aid in this process. Chuah and Roth [7] extended Foley et. al.'s user interaction framework [8] by incorporating BVI (Basic Visualization Interactions), which is a more detailed characterization of data filters in the context of information visualization. Tweedie [15] presented a data transformation model similar to Lee and Grinstein's model [12], an interactivity model that classifies the interactions based on the amount of control the user has over the process, and a state model similar to Chuah and Roth's model [7].

The reason for moving toward Data State Models is that, in information visualization especially, we are often exploring the relationship between view and value. Because values are often represented abstractly by views, the loose coupling between view/value forces visualizers to think harder about the states of the data values.

Because of the different emphasis in expression, the two models have resulted in very different user interfaces. They result in different characteristics when utilized to implement a visualization system. (a) Data Flow Model based systems create modules that correspond to the process nodes, and data transferring mechanisms are created to connect the modules. Typically screen real estate is devoted to representing these modules. (b) On the other hand, Data State Model based system create data stores that correspond to the data states, with data processing procedures created to connect the data states. Typically the display is devoted to presenting these data states with visualizations. These arguments suggest that each model has its own strengths and weaknesses. Data State Models may represent the visualization states better than Data Flow Models, while Data Flow Models are superior in representing the visualization process. A potential happy medium would be to provide for a way to move fluidly between the two alternative models, and that the two alternative methods may support one another.

Therefore, we need better understanding of a major question: "Whether Data State Models are as expressive as Data Flow Models?" This paper seeks to compare the two models by first proving that Data State Models can represent anything that is expressed by the Data Flow Model.

### 3. EXPRESSIVENESS OF BOTH VISUALIZATION MODELS

#### 3.1 Expanding the Data Flow Model

The Data Flow Model uses nodes to represent processes because the model focuses on the data transformations. The edges represent the flow of the data from one process stage to the next. An edge

exists only if a process transforms the data into a new form. Often, in data flow networks, the data state is not explicitly represented by distinct edges. In other words, the edges represent data *stages* instead of *states*. For example, consider a single data flow chart that constructs a scatter plot (raw data set  $\rightarrow$  extract point set  $\rightarrow$  create scatter plot  $\rightarrow$  view). Consider applying two different data sets to this flow chart. Since the model does not capture data states, these two different data set can flow down the same pipes, even though a different data set clearly represents a different data state. In the Data Flow Model, since the format of the data does not change, there is neither new edges nor new nodes to represent this.

In order for the Data Flow Model to capture the same amount of detail as the Data State Model, we first define a canonical form of the Data Flow Model that insists on having each edge representing only a single distinct data state. The implication of this is that we force the Data Flow Model to capture more details of the visualization process. The syntax and semantics of the model remain the same. However, this change does not fundamentally change the Data Flow Model, as it still shows how data flows through the system and how processes and algorithms transform the data. The model still shows the functional dependencies between the processes. This definition simply expands the Data Flow Model based on data set instances.

**Definition 1** *The canonical form of the Data Flow Model restricts each edge to represent only a single data state.*

Below, we will show that the canonical form of the Data Flow Model is just as expressive as the Data State Model. By expanding the notion of data flow network in this way, we can then show the equivalence of the Data Flow Model and the Data State Model.

### 3.2 Visualization Equivalence

In this section, we first introduce the notion of equivalence between two visualization models by defining the idea of expressiveness of the visualization model. Then we use a duality transformation to show that one model can be transformed into the other and generate the correct output.

To start, we first ask the question, “what is a visualization model?” A visualization model describes a visualization process, which is a transformation process. A transformation process is composed of a series of transformation steps. So we first define the visualization transformation. We will use a mathematical functional description:

**Definition 2** *A visualization transformation or visualization operator  $n$  processes information  $d$  from a domain  $D$  and maps it into information  $d'$  in a different domain  $D'$ .  $n(d) = d'$ , where  $d \in D$  and  $d' \in D'$ .*

The domain specifies the structure of the information. (a) A domain may be a *visualization domain*, which means that the elements in that domain are mapped or easily mappable onto the display screen. (b) A domain may be a *data domain*, which means the elements in that domain are raw data that are not yet mapped onto the display screen. A domain may be as simple as the set of natural numbers, or as complex as database records of user transactions with a hypertext system. For example, a genetic sequence similarity alignment record is a single element in the genetic sequence similarity domain.

**Definition 3** *A visualization function  $m$  is a visualization transformation that maps from a data domain  $D$  to a visualization domain  $V$ . So given  $d \in D$ ,  $m(d) = v$ , where  $v \in V$ . Typically,*

*$m$  is composed of a series of visualization transformations from a set of transformations  $N = n_1, n_2, \dots$ . Each transformation  $n_i$  in a series maps from a domain  $D_i$  to  $D'_i$ . The domain of the next transformation in the series must be the same as  $D'_i$ . For example, if  $m_3 = (n_2, n_3, n_2, n_1)$ , then applying  $m_3$  to  $d$  gives  $m_3(d) = n_1(n_2(n_3(n_2(d))))$ , where  $d \in D_2$ ,  $n_2(d) \in D'_2 = D_3$ ,  $n_3(n_2(d)) \in D'_3 = D_2$ , and so on.*

The output of a visualization transformation may not be compatible with the input of another visualization transformation. This is because each visualization transformation takes a specific kind of data from a domain as input. Therefore, not all visualization transformations can be composed together, and we cannot simply construct a composition of an arbitrary series of visualization transformations. The input domain of a transformation must be the same as the output domain of another transformation for the two transformations to be connected in a series.

**Definition 4** *A visualization model  $M$  is a set of visualization functions  $M = m_1, m_2, m_3, \dots$*

An **instance** of a visualization model is a particular application, case, or example of the visualization model. The act of constructing a case is called **instantiation**.

**Definition 5** *In the definition above, each  $m_i$  is an instance of  $M$ , and  $m_i(d)$  is an instantiation of  $M$ .*

A model is a series of transformations, which can be viewed as a mapping function from data to view. An instance is a particular defined series of transformation designed for a particular type of data. Next, we define the relation “as-expressive-as”:

**Definition 6** *We say a visualization model  $A$  is as-expressive-as another model  $B$  if given an instance  $b$  of  $B$ , we can find a model  $a$  of  $A$  such that for all inputs  $i$ ,  $a(i)$  gives exactly the same output as  $b(i)$ . (That is, given  $b \in B$  and for all inputs  $i$ ,  $\exists a \in A$  such that  $a(i) = b(i)$ .)*

Notice that expressiveness is not a symmetric relation ( $A$  being as-expressive-as  $B$  does not mean  $B$  is as-expressive-as  $A$ ). Next, we use the concept of antisymmetric relation to define the meaning of equivalence between two visualization models. A relation  $R$  is antisymmetric if for all  $x$  and  $y$ ,  $xRy$  and  $yRx$  implies  $x = y$ .

**Definition 7** *We say that two visualization models are equivalent in expressiveness if and only if  $A$  is as-expressive-as  $B$  and  $B$  is as-expressive-as  $A$ .*

Given this definition, we wish to show:

**Theorem 1** *The Data State Model is equivalent in expressiveness to the Data Flow Model.*

**Proof:** We will prove this by construction. Using a principle we call *Duality*, we will show that given an instance of the Data Flow Model we can construct a Data State Model that gives exactly the same output and vice versa. To prove this in both directions, we use the directed graph expression of both models and the duality transformation. Note that the Data Flow Model first must be converted into the canonical form. The transformation outlined below only works on canonical Data Flow Models.

Using Data Flow Model notations, an instance  $m_i$  can be expressed as a path  $p = n_i \xrightarrow{d_i} n_j \xrightarrow{d_j} n_k \xrightarrow{d_k} \dots$  through a directed graph where the nodes are the transformation steps  $N =$

$n_i, n_j, n_k, \dots$  and the data states  $d_i, d_j, d_k, \dots$  are the edges. Using the Data State Model notation, the same instance can be expressed as  $p = d_i \xrightarrow{n_i} d_j \xrightarrow{n_j} d_k \xrightarrow{n_k} \dots$ , where the nodes are the data states  $d_i, d_j, d_k, \dots$  and the edges are the transformation steps  $N = n_i, n_j, n_k, \dots$ .

#### Duality Transformation

To perform the duality transformation<sup>1</sup>, simply take each edge in the model and convert it into a node and convert each node into edges. Mathematically, given a model  $G = (V, E)$ , the duality transformation constructs  $D(G) = G' = (V', E')$  where:

- For each edge  $e$  in  $E(G)$ , we construct a vertex  $v'_e$  in  $V'$ , and
- For each node  $v$  in  $V(G)$ , for each pair of edges  $(e_1, e_2)$ , where  $e_1$  goes into  $v$ , and  $e_2$  exits  $v$ , we construct an edge  $e'$  from  $v'_{e_1}$  to  $v'_{e_2}$ , where  $v'_{e_1}$  and  $v'_{e_2}$  are the corresponding vertices of  $e_1$  and  $e_2$  respectively in  $G'$ .

Note that in this transformation, a single node may have several corresponding dual edges. Figure 2 shows an example of this transformation.

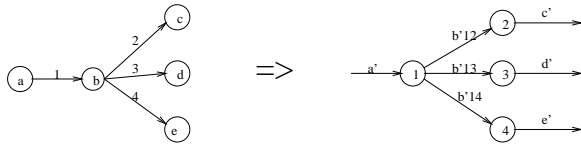


Figure 2: An example of the Duality Transformation

The duality transformation switches the role of the nodes and edges. For example, in one direction of this application, our state model  $G = (V, E)$  uses nodes to represent data states, and edges to represent distinct processes. ( $V = \text{set of data states}$ ,  $E = \text{set of transformation processes}$ ). After the duality transformation, we obtain a  $D(G)$  that has  $V' = \text{set of transformation processes}$  and  $E' = \text{set of data states}$ .

To finish the proof using this duality transformation, we need to show that given a path  $p$  in  $G$ , we have a path  $p'$  in  $G'$  that produces the same output as  $p$ . Say  $p = v_1 \xrightarrow{e_1} v_2 \xrightarrow{e_2} \dots \xrightarrow{e_{i-1}} v_i$ , the equivalent path  $p'$  is  $p' = v'_1 \xrightarrow{e'_1} v'_2 \xrightarrow{e'_2} \dots \xrightarrow{e'_i}$ . The crucial realization is that a vertex  $v_j$  in  $p$  has its corresponding edge in  $p'$  as  $e'_j = (v'_{e_{j-1}}, v'_{e_j})$ . Using Figure 2 as an example, the path  $(a, 1, b, 2, c)$  is transformed into  $(a', 1', b'_{12}, 2', c')$ .

So if  $G$  uses the Data State Model and  $G'$  uses the Data Flow Model,  $p$  in the functional notation gives  $e_{i-1}(\dots e_2(e_1(v_1))) = v_i$ , and  $p'$  gives  $v'_{i-1}(\dots v'_2(v'_1(e'_1))) = e'_i$ . Given that the  $v$ s in  $G$  are states that correspond to  $e'$ s in  $G'$ , we see that the inputs  $v_1 = e'_1$  and  $v_i = e'_i$  and the two paths gives same output. In the reverse case where  $G$  is the Data Flow Model and  $G'$  is the Data State Model, we can obtain the same output by the same principle. This is because the transformation shows that the order of processes and data states that  $p$  visits have exact equivalents in  $p'$ . Therefore, using this duality transformation, we can construct a data flow model that is *as-expressive-as* a given data state model, and vice versa. Hence, the two models are *equivalent in expressiveness*.

## 4. CONCLUSION

<sup>1</sup>This is not the same as the *dual graph* in graph theory. The dual graph  $dual(G)$  of a graph  $G$  constructs a node for each enclosed region. If two regions share an edge, then we construct an edge between the corresponding vertices in  $dual(G)$ .

Visualization can be viewed as a series of transformations that transcribes data values into graphical views. The relationship between view and value has been modeled primarily in two graphical models: Data Flow Model and Data State Model. The Data Flow Model has been well-established, especially in scientific visualization, and its capabilities and expressiveness is well-understood.

Our main concern in this paper is to understand and compare the two models. We proved that the Data State Model is equivalent in visualization expressiveness to the Data Flow Model, and vice versa, which means that we can model the same visualizations using either model. Even though the two models are equivalent in expressiveness, their instantiation in the user interface gives each model different strengths and weaknesses. We hope this paper have pointed some directions for future researchers to further understand the differences between the two models.

## Acknowledgement

The author wishes to thank John Riedl for discussions.

## 5. REFERENCES

- [1] Advanced Visualization System home page. <http://www.avis.com>, February 1999.
- [2] Jacques Bertin. *Graphics and Graphic Information Processing*. Walter de Gruyter, 1981.
- [3] Ed H. Chi. A taxonomy of visualization techniques using the data state reference model. In *Proc. of the Symposium on Information Visualization (InfoVis '00)*, pages 69–75. IEEE Press, 2000. Salt Lake City, Utah.
- [4] Ed H. Chi, John Riedl, Phillip Barry, and Joseph Konstan. Principles for information visualization spreadsheets. *IEEE Computer Graphics and Applications (Special Issue on Visualization)*, pages 30–38, July 1998.
- [5] Ed H. Chi and John T. Riedl. An operator interaction framework for visualization systems. In *Proceedings of the Symposium on Information Visualization '98*, pages 63–70. IEEE CS, October 1998. Research Triangle Park, North Carolina.
- [6] Ed Huai-hsin Chi, Phillip Barry, John Riedl, and Joseph Konstan. A spreadsheet approach to information visualization. In *Proceedings of the Symposium on Information Visualization '97*, pages 17–24, 116. IEEE CS, 1997. Phoenix, Arizona.
- [7] Mei C. Chuah and Steven F. Roth. On the semantics of interactive visualization. In *Proceedings of the Symposium on Information Visualization '96*, pages 29–36. IEEE CS, 1996. San Francisco, California.
- [8] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990.
- [9] Paul E. Haeberli. ConMan: A visual programming language for interactive graphics. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 103–111, August 1988.
- [10] IBM Visualization Data Explorer (DX). <http://www.almaden.ibm.com/dx/>, February 1999.
- [11] IRIS Explorer home page. <http://www.nag.co.uk/Welcome.IEC.html>, February 1999.
- [12] John Peter Lee and Georges G. Grinstein. An architecture for retaining and analyzing visual explorations of databases. In *Proc. IEEE Visualization '95*, pages 101–108. IEEE CS, 1995.
- [13] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Prentice Hall, 1996.
- [14] Silicon Graphics Computer Systems. *IRIS Explorer User's Guide*. Silicon Graphics Computer Systems, Mountain View, CA, 1991.
- [15] Lisa Tweedie. Characterizing interactive externalizations. In *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1, pages 375–382, 1997.
- [16] Craig Upson, Jr. Thomas Faulhaber, David Kamins, David Laidlaw, David Schlegel, Jeffery Vroom, Robert Gurwitz, and Andries van Dam. The Application Visualization System: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, pages 30–42, July 1989.