On the Semantics of Interactive Visualizations

Mei C. Chuah, Steven F. Roth
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, 15 213, USA
Tel: 1-412-268-2145
E-mail: {mei+, roth}@cs.cmu.edu

Abstract

Interactive techniques are powerful tools for manipulating visualizations to analyze, communicate and acquire information. This is especially true for large data sets or complex 3D visualizations. Although many new types of interaction have been introduced recently, very little work has been done on understanding what their components are, how they are related and how they can be combined. This paper begins to address these issues with a framework for classifying interactive visualizations. Our goal is a framework that will enable us to develop toolkits for assembling visualization interfaces both interactively and automatically.

Keywords: Information visualization, interactive techniques, user interfaces, automatic presentation systems, graphics.

1. Introduction

In the last few years many interactive techniques and metaphors have been introduced for different visualization types. These techniques allow users to deform space[6], deform objects[1], view objects at different levels of abstraction[3] and a wide variety of other functions. Although interactive techniques facilitate the use of visualizations, they have been implemented and combined as point solutions to focused problems. No unifying framework exists that describes the structure of individual techniques and how they can be combined. As a result it is difficult to extract the critical functionality contributed by different techniques in a system, making it difficult to apply its design features in other systems. We propose a framework for decomposing visualization system user interfaces into primitive interactive components, describe a functional classification of the different primitives, and present rules for composing and structuring these primitives. The goal of the framework is to enable us to compare systems and reuse previous design elements, as well as to guide our design of a toolkit for constructing visualization interfaces through the composition of interaction primitives. This infrastructure is the first step towards developing an automatic presentation system that can generate interactive visualizations.

To begin our characterization of interactive visualization techniques, we expand on previous work done on user interfaces[2,5,7]. Foley et al. [2] described three levels of design for interfaces: *lexical design*, *syntactic design* and *semantic design*. *Lexical design* refers to how input and output primitives are derived from basic hardware functions. Input primitives include all physical devices while output primitives could take the form of visual, audio or sensory elements. In this paper we are only concerned with visual output. *Syntactic design* consists of a set of rules by which primitive input or output units can be composed or joined to form ordered sequences of inputs and outputs. For example, the mouse movements, clicks and releases required to specify a bounding box together define a focus area in an interface but do not define its meaning (i.e. its function). *Semantic design* defines the meaning of a sequence of actions as a *task*. For example, mouse clicks, bounding boxes and sliders can all be used to define a *selection* of objects. In this case the meaning is the selection task, while the actions used to achieve the task could take multiple syntactic forms. Similarly, an action can have several meanings. For example a bounding box can be used for selection, aggregation, deletion, copy, or other functions.

Figure 1 shows the three levels of design described above. The gray circles indicate the components added to Foley et al.'s framework in order to deal with interactive information visualizations. This paper focuses only on the semantic level. There is much previous work on the lexical and syntactic levels [2,5,7] and we can incorporate it into our framework.

The semantic primitive in our framework is the *basic visualization interaction* (BVI). Understanding a BVI requires knowing its inputs (whether provided by a user through an input method or fixed by the interface designer). It also requires understanding its effects on the graphical, data, and control states of the visualization user interface environment. Our framework also characterizes the composition of BVIs into application interface functions that we normally see in visualization systems such as aggregation, filtering, sort, and coordinated displays. Most visualization applications consist of combinations of multiple BVIs for one or more visualizations. For example, in the Table Lens[8], users perform multiple tasks on a table visualization. Users manipulate a lens to focus attention and sort table rows with gestures. These provide *magnify* and *sort* tasks. The magnify task can be decomposed into a *set-creation* BVI to define the graphical rows to be magnified and two *set-graphical-value* BVIs to increase the size of the focus rows and decrease the context rows.

Other systems provide interface operations for multiple coordinated visualizations. For example, in the SAGE system[9], painting, dynamic query and aggregation operations can affect multiple active visualizations simultaneously. Each of these operations are compositions of multiple BVIs, which we illustrate in the next sections.

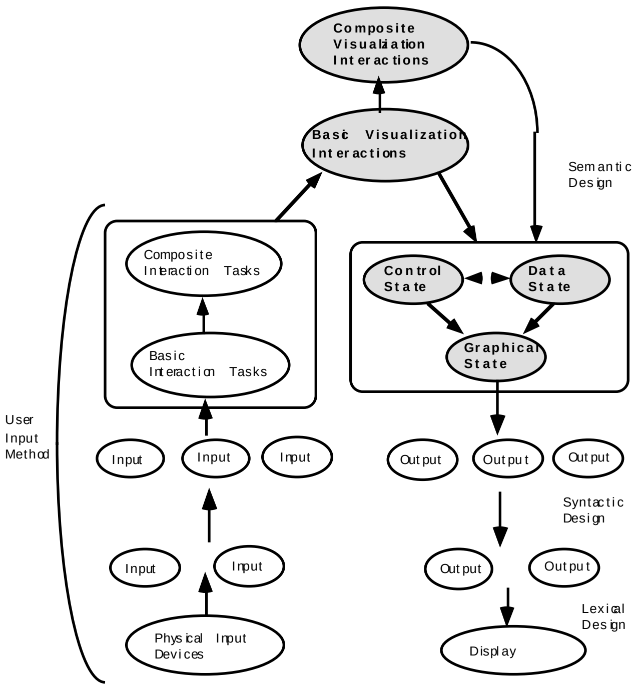

Figure 1: Expanded interface architecture for visualization interaction

In this paper:

- We propose a set of basic visualization interaction primitives and describe their inputs and effects on the state of a visualization system. This specifies the flows into and out of sequences of BVIs. We will give some examples of the relation between BVIs and the physical and virtual devices that can be linked to them to provide inputs.
- We propose a preliminary classification of BVIs as a starting point for understanding the kind of framework we believe is needed to understand their expressiveness and to use them to compare systems functionally. It also identifies the types of building blocks we might provide in an interaction toolkit. Once we extend this framework to include effectiveness and user task information we can determine which techniques or combination of techniques will best serve a given set of user tasks.
- We describe issues involved in composing BVIs to form composite visualization interactions. Composition is a critical function in a toolkit and in automated design because it opens up the design space of possible interactive behaviors. By combining BVI's, we can develop complex behaviors and effects.

2. Real-estate example

In this section we present a simple interface for examining a data base of house sales. Figure 2 shows the locations of houses on a map and a dynamic query slider that controls their visibility based on *asking-price*. Selection occurs by clicking on houses or enclosing them with a bounding box. Selection can be performed in conjunction with filtering (dynamic query slider) to choose houses based on both their *asking-price* and their locations. For example, a user could make visible those houses costing more than $100K and select only those in the southern region using the bounding box (turning them red):

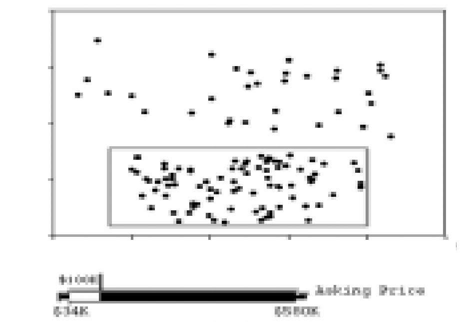

Figure 2: Map display of house locations

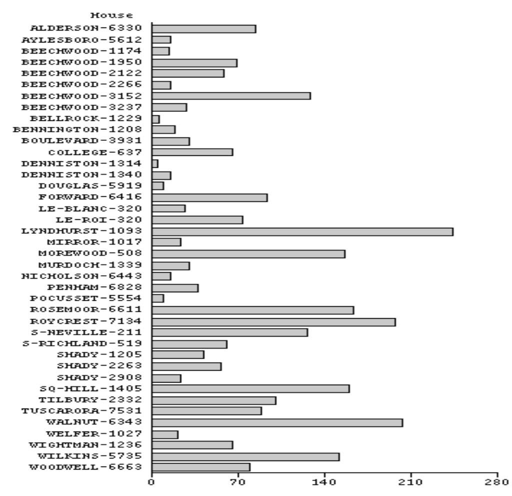

Figure 3a: Bar chart showing days on the market for subset of houses

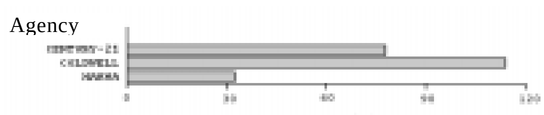

Figure 3b: Houses in Figure 3a aggregated by Real Estate agency.

After selecting the houses of interest, a user may want to know how quickly they were sold by creating a new graphic displaying this information. The user *drags* the selected elements to an empty display with *addresses* and *days-on-the-market* on the axes, which causes the elements to be visualized as bars (Figure 3a). In order to compare the speed with which agencies sold these houses, the user then aggregates the houses by *agency,* replacing the individual houses with three agency averages (Figure 3b). Finally, an agency aggregate bar can be *painted green* to change the color of corresponding houses on the map (Figure 1). This would show regions where the painted agency sold its houses.

The example illustrates basic operations commonly performed in data analysis: filtering, selection, aggregation, coordinated highlighting, and creating graphics. We also want to emphasize that while each of the functions achieve different effects, they also share similar component basic interactions. For example, several operations involved creating or changing elements of a set. These operations include filtering with a slider, selecting with a bounding box, dragging a group of objects into a new display, aggregating, and coordinated painting across displays. Each also involved changing the graphical properties of the elements. For example, filtering with the query slider changed visibility, painting changed color, aggregation deleted graphical objects of elements and added aggregate objects.

Our goal, then, is to define a set of basic visualization interactions like *create-set* and *set-graphical-value* that underlie these application tasks. The next section describes BVIs, their inputs and outputs (effects).

3. The basic visualization interaction (BVI)

A BVI is fully described only when we specify the BVI inputs, outputs and operation. The number and type of inputs required depend on the BVI. Inputs can be set by a user via an interface (a *user input method*), or they can also be predefined by the designer (a *default specification*). A user input method consists of combinations of device inputs and basic and composite interaction tasks (refer to Figure 1).

BVI inputs pass through a BVI operation which consists of two phases: condition check and action. The condition check determines whether all the required inputs and values are sufficient for the operation to occur. The proceeding action will then change the graphical, data or control state of the visualization interface. An action may also generate outputs that serve as inputs for other BVIs, thereby producing sequences that can potentially achieve complex tasks.

3.1 Basic visualization interaction inputs

The *inputs* provide ways in which users or designers affect BVI behavior. The dynamic query slider in our real-estate example, has as inputs: a data *attribute*, an initial visible set (*control object*), two *values* and a focus set which includes all elements in the visualization. Changing these inputs will cause an *express-membership* BVI to calculate a new definition of the members of the initial visible set (i.e. the intention defined by the slider). There are five classes of input arguments. We describe each and briefly outline how they can be derived from both physical and virtual input devices based on the framework described in [7] as well as interaction tasks described in [2]. In the right of Figure 4, we show the input classes BVIs require.

Attribute: There are three attribute types: graphic (e.g., *color*, *shape*, and *x-position)*, data (e.g., *house address*, *house price*, and *date-sold*), and state (e.g., *number of graphical elements*). Attributes are chosen by users using the *selection* basic interaction task, defined in [2], as the task of choosing an element from a choice set. This can be achieved through naming or menus. Each of these techniques can be achieved through different input device classes. For example, naming can be achieved with a 1-dimensional discrete position device (e.g., keys) and menus require a device(s) that is capable of expressing a (2,3)-dimensional continuous position and a 1-dimensional discrete position (e.g., a mouse). More details on device expressiveness can be found in [7].

Control Object: Control objects or reference objects may be a data object, a graphic object, a virtual device object (e.g., handle, slider) or a set object. Control objects are used to provide a point of reference for the BVI. For example, adding new members into a set requires a control object that defines the reference set into which we add members. In our dynamic query slider example, the visible set is the *control object*. Defining control objects is also done through the *selection* basic interaction task. In order to do this we could select the names of the objects, similar to the way in which attributes were selected. Alternatively, we could select the graphical representations of the reference objects. The latter can be achieved by the pointing technique. Pointing requires a (2,3)-dimensional continuous position device (e.g., mouse moves).

Value: Based on the framework specified in [7], there can be two different classes of values: linear and rotary. This distinction is important for choosing input devices based on their effectiveness. For example, a dial input device is more effective for specifying rotary values than a slider. Linear inputs consist of position, movement, force, and delta-force. Rotary inputs consist of rotation, delta rotation, torque, and delta torque. Which type of value is needed depends on both the BVI and the feedback method. Values are defined through the *quantify* basic interaction task which specifies a value between some minimum and maximum. In the dynamic query slider example, two linear values are provided by the slider input device method.

Formula: A formula defines relationships among multiple variables. In our framework it is used to specify an effect on specific variables based on the values of other variables. Some example formulas used in our framework include the object encoding formula which specifies which graphical representation to use for data objects based on their characteristics. Distortion formulas are another example.

They specify how the positional parameters of an object or an area should be altered based on a cost of value function or according to a reference object. Formulas are usually predefined. However, they may also be specified by users.

Focus Set: The focus set argument defines the set of entities on which a BVI operates. The focus can either be defined through user input methods, assigned to a predefined set, or output by another BVI. User input methods can either define single objects with a single (2,3)-dimension positional input device (e.g., mouse move), or an object group and area with multiple (2,3)-dimension positional values (e.g., bounding box). The dynamic query slider in our real-estate example has a predefined focus set containing all the objects on the map (Figure 1).

A focus set may consist of two different entity types: objects or space. An object usually refers to the graphical representation of a particular data point (e.g. point, line, node, link, axis). A data object may be represented multiple times in several visualizations. An operation may affect the graphical representations of data objects or it may affect the data objects themselves. The other entity type is space. Space selection may be of type area or volume. Areas are defined for both two-dimensional and three-dimensional visualizations. Spaces and objects can be operated on singly, as groups or universally (i.e. on the entire visualization).

It is important to distinguish between space and object type entities because the same operation, when applied to an object or a space, may result in very different effects. For example, magnification of an object will only cause the objects to grow while the surrounding areas remain unchanged. The advantage of this method is that the absolute position of objects remain constant. Area magnification causes not only the objects within the area to expand, but the space between the objects to expand as well. Unlike the previous case, object positions are no longer static. As the area is magnified, objects move farther apart. Hollands reported in [4] that this effect may be disorienting to the user. However, the advantage of area magnification is that no matter how large objects get they will not overlap, because the surrounding space expands proportionately. This property does not hold for object magnification. Such distinctions have significant implications for the effectiveness of a technique, and on the tasks it can support.

3.2 Basic visualization interaction outputs

BVI outputs may affect the graphical, data or control state of the system. Different BVIs have different output/effect choices. For example the *set-graphical-value* BVI may change any attribute of the graphical objects in the visualization. Its effect choices however are limited to graphical objects. The *derive-attributes* BVI on the other hand can only affect data objects. For a particular BVI, different output methods may require different inputs. For example, suppose we have a *set-graphical-value* BVI. If the output method is orientation of the focus set objects, then a rotation input value might be needed. If a positional change is desired, then a linear input value might be more appropriate. Furthermore, appropriateness of output is strongly dependent on user task. For example, displacing position brings out occluded objects , but displacing color does not.

Graphical State: The graphical state refers to all objects that are currently visualized. This includes graphical objects as well as axes, labels, and keys. Changes in the graphical state can be made to objects (e.g. changing from marks to bars) or to attributes (e.g. color, transparency, visibility). There are two types of graphical attributes: *spatial* (e.g. size, shape, orientation and position) and *surface* properties (e.g. color, transparency, blinking, texture.

In the real-estate example, coordinated painting involves coloring interesting objects green. This is specified by the designer through a default specification, which could be changed to blue if desired. Similarly, a designer may change the highlight attribute from color to visibility, so only interesting objects are visible. In the same way, a dynamic query slider can control size or color instead of visibility.

Data State: The data state contains information about all of the data elements (e.g. *house*-1, *house*-2) that are currently in the system. This includes their attributes (e.g. *asking-price*, *days-on-market*) and attribute characterizations (e.g. *cardinality, data-type*). Changes in the data state occur as new data is read in, or as users delete and change existing data elements.

Control State: The control state contains internal information generated during operation of the system. It includes four information types: virtual objects, global properties, interaction state and history. Virtual objects are abstractions created by the system during the course of interaction (e.g. set abstractions). Global properties describe general system state variables (e.g. visualizations currently open, current directory, access permissions). Interaction state refers to BVIs used in a system. This includes information about interaction constraints and associations between BVIs and visualizations. Finally history information includes traces of user activity and previous user errors.

Information in the three states are needed to describe the function of BVIs. BVIs often query the system state before invoking their operations. For example, an interactive technique may use visibility for filtering when the total number of objects in the visualization is large and highlighting when the number of elements is small. Coordinated painting in our real estate example also queries the system state. When a user clicks on the *agency* aggregate bars, the control state is queried in order to retrieve all of the house names associated with the selected aggregates. These house names are then used to construct the highlighted set on the map.

Note that changes to the control and data states do not provide any feedback to the user because their effects are internal. Changes to the graphical state may provide user feedback, simply because their outputs are very noticeable. The dynamic query slider in our example, changes object visibility, which gives sufficient user feedback. In other interactions, (e.g. the scaling operation in SDM [1]), large changes to the scale may sometimes only cause small changes to the selected objects, so feedback mechanisms must be included about the occurrence and effects of an action.

4. BVI operation classification

There are three BVI classes: *graphical operations*, which change the appearance of visualizations, *data operations* which manipulate data encoded in visualizations, and *set operations* which create and manipulate object sets. Data objects are only mapped to graphical objects, not spaces, so data operations will not refer to space entities.

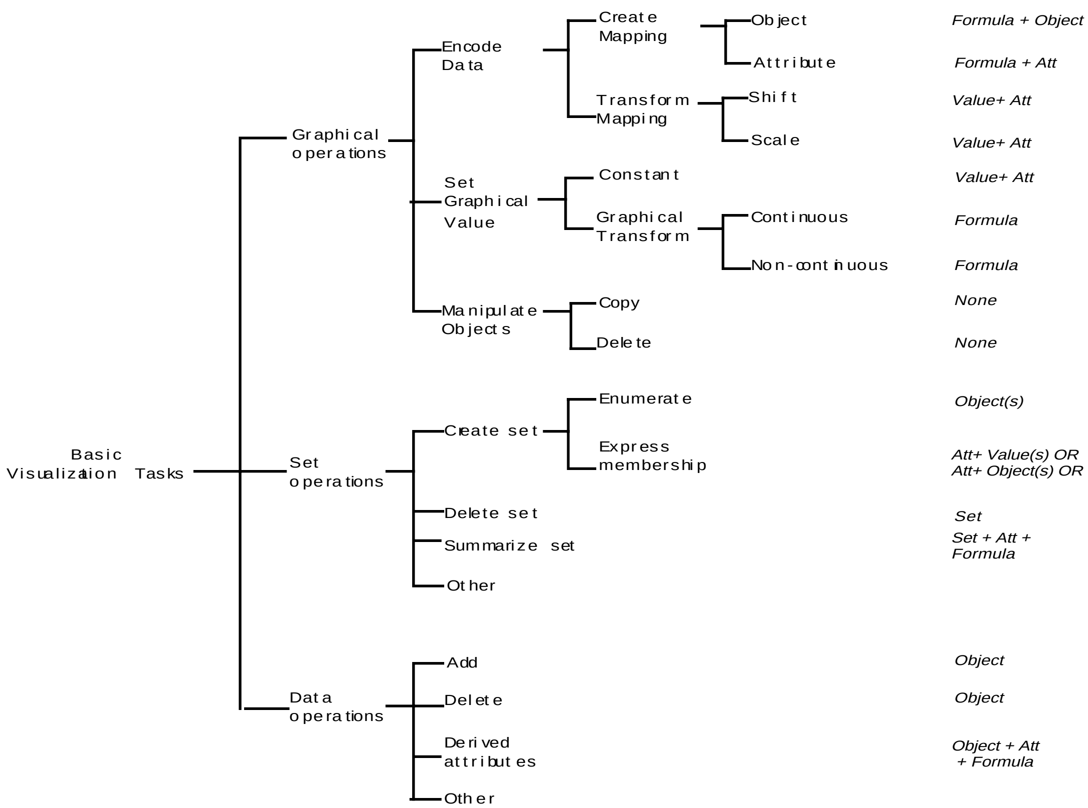

Figure 4: Basic Visualization Interaction classification hierarchy

Each of these classes affect different output states in the system. Graphical operations affect the graphical representations of data objects (i.e. the graphical state). Set operations affect the control state and data operations affect the data state. Changing the control and data state could however, cause secondary effects to the graphical state. For example, deleting a data object will cause the internal database to change. In addition, it also causes all of the graphical representations of the deleted data object to be removed, which changes the graphical state.

Figure 4 shows a tree of the three BVI classes. The leaves of the tree indicate the different types of BVI. The italicized text towards the right of the tree show the input types needed by each of the BVIs. For example, a *shift* BVI requires an attribute (which could be an attribute in the graphical, data or control state) and a value. In the interest of space, only a partial tree is shown and details about input types (e.g., position, force, rotation and torque) are not included. In addition, the *other* component in both the set and data subtree indicate that there are unlisted operations within those classes. It is not our intention here to capture all of the possible techniques but rather to describe a basic set of operations that capture most of the interactive behaviors that exist today.

Graphical Operations: Graphical operations can be divided into *encode-data*, *set-graphical-value*, and *manipulate-objects*. *Encode-data* refers to operations that change or transform mappings between data and their graphical representations. The *change-mapping* operation is achieved by altering graphical *object* or graphical *attribute* mappings. An example of the former is switching from points to bars to represent attributes of houses. The drag-and-drop operation in Visage [10] is an instance of this class of interaction. An example of the latter is changing the encoding of *house-price* from the size of a point to saturation.

Mappings between graphical attributes and data can also be transformed, either by *shifting* the encoding range or *rescaling* the encoding range. *Scale* operations are usually used to magnify differences among values of a particular attribute (e.g. scale height), and *shift* operations are used to separate out sets of objects (e.g. shifting x-position). Examples of *shift* and *scale* operations can be found in the SDM system[1].

As shown in Figure 4, operations that change graphical representations can also be of type *set-graphical-value*. These operations alter the visual representations of selected entities uniformly by simply setting the values to a *constant* or according to a formula (*graphical- transform*). Painting a set of objects red is an example of an operation that sets the value of the color attribute to a constant (red). The formulas available for transforming graphical values have been classified by Leung [6] into continuous and non-continuous functions. The difference between *set-graphical-value* operations and the *encode-data* operations is that for the latter, the altered graphical attributes must encode a data attribute, whereas in the former, this need not be the case. Since *set-graphical-value* operations are not related to the underlying data, they can be applied equally well to both object and space type entities.

The third class of graphical operations are for manipulating graphical objects, including creating and deleting them. Unlike the *encode-data* and *set-graphical-value* operations, these operations do not change graphical attributes nor do they change the mappings between graphical objects and data. Instead, they operate on the graphical objects as a unit of manipulation.

Set operations: Set operations refer to all those operations that act on or form sets. These operations include creating sets, deleting sets, summarizing sets, joining sets, intersecting sets and so forth. Sets provide a way for users to expand the underlying data with new classification information. For example, the aggregation task in our real-estate example caused the creation of multiple sets to represent the classification of houses by *agency*. When the user aggregated the houses based on *agency* a set was created for all the houses sold by each agency. An object was created for each set and then visualized.

Sets are populated by enumerating members of the set or by expressing conditions for set membership. Enumerating set members is achieved by having the user explicitly pick members from the visualization, by having the designer define the sets apriori or by getting the objects from the system control state. For example, during the coordinated highlighting task in our real-estate example, the highlighted set on the map is constructed based on the enumeration of data-objects within the selected agency aggregates (these objects are obtained by querying the control state).

Set membership is defined through a formula or a constraint, which may be dynamically altered. Elements that fulfill the set constraint automatically get added or removed. The dynamic query slider technique in our example consists of a set associated with a membership constraint. The slider controls this constraint and cause elements to be dynamically added to or removed from the set when the slider is moved.

Sets have group characteristics. Members that join the set will automatically inherit those group characteristics. Upon leaving the group, members will lose those characteristics and revert back to their individual characteristics. For example in the real-estate filtering task, the slider value controls set membership. When elements enter the set, they become visible because they inherit the visibility property of the set. When they leave the set, they revert back to their individual visibility value which is off. In the same example, a set is created with a bounding box and is painted red. Dragging one element of the set, causes the position of the set to change. This causes the position of all elements within the set to change as well.

Data operations: Data operations affect the data elements contained within the visualization. Data operations are especially useful in creating simulations or carrying out what-if analyses. This can be done by changing sets of data values and then observing what changes these cause to the other data values. Another useful data operation is deriving new attributes for the data objects. During analysis, users usually discover new facts about the data and it is useful to be able to augment the data with new findings.

5. Composite visualization interaction

To support complex tasks, BVIs can be combined to form composites. There are three types of composition: *independent composition*, *set composition* and *chained composition*. In *independent composition*, the BVIs are made available for the same visualization but are not related except when there are conflicts in desired effects. Effects are applied orthogonally and operations can be executed in any order or in parallel. Each operation has its own user input methods. In the real-estate example, the map display has a selection interaction with a bounding box and a filtering interaction with a slider. These operations are independently composed into the same visualization. Each operation has separate and distinct input methods (bounding box and slider). Elements that are filtered and therefore not visible cannot be selected with a bounding box. The semantics of items that are selected, made invisible and then visible again are potentially confusing (e.g. when they are invisible, do they remain in the selected set?). Our characterization enables the designer to represent the alternatives.

In *set composition*, operations are grouped so that when one is triggered, the others get executed as well. As before, effects are applied orthogonally and operations can be executed in any order . Operations that are composed all affect the same focus set. In our real-estate example, coordinated painting consists of a *set-graphical-value* BVI that changes the color of the selected objects to a constant (green). This BVI is set composed with an *enumerate-set* BVI. In operation, either the *set-graphical-value* BVI is invoked first to paint a bar in Figure 3b green, or, the highlight set for the visualization in Figure 1 is defined by the *enumerate-set* BVI. The order in which these operations occur do not affect the final result of the visualizations. Note also that both operations used the same focus object(s), which in this case were the selected agency aggregate bars. The *set-graphical-value* BVI colored the agency bars green while the *enumerate-set* BVI used the agency bars to form a new green highlighted set in Figure 1.

This composite BVI can be tied to an input device or multiple input devices. For example, suppose the composite BVI is tied to a bounding box and a mouse click. When a user clicks on a bar, it will be highlighted and all related houses in Figure 1 will be highlighted as well. This will also occur to the set of objects chosen by a bounding box. Note that this is different from the previous composition where each BVI has separate and distinct input devices. Here, all BVIs within the set are tied to the same input method.

Finally, BVIs may be sequenced or chained. In *chained composition* a particular BVI has its focus set defined by the previous BVI in the chain. For example, the aggregate by *agency* task performed in the real-estate example, requires several *create-set* BVIs (one for each agency-type) to be set composed. To display objects representing the newly created sets, these BVIs must be chain composed to the *create-graphical-object* BVI, which creates representations of the sets and adds them to a visualization. The *create-graphical-object* BVI has to receive its focus set from the *create-set* BVIs because the objects to be visualized (i.e. the sets) only exist after the *create-set* BVIs have been executed. The aggregation operation actually replaces the detailed objects with aggregates - bars for individual houses are deleted as aggregates are created. We can represent this as set composing the *delete-object* BVI with the previous composite for creating an aggregate

.

Different BVIs in the chain may have different input methods. When a BVI in a chain is triggered, it gets executed and then all other BVIs that come after it get executed in turn. BVIs that come before it, however, do not get executed.

An issue that arises in composition is *resource conflicts*. *Resource conflicts* may occur when two BVIs try to change the same resource (e.g., data, graphic or state attribute). For example, in the real-estate scenario, the map visualization could have two different types of sets, a set painted red using the bounding box, and a coordinated set painted green using the agency display. When an object falls within both sets, there is uncertainty as to which color it should be painted. In addition, if both interactions highlighted objects to be red then there is ambiguity as to how/why a particular object got highlighted. Dealing with collisions in interactions is a complicated process because it depends on the visualization, the operation, and the task. If the task does not require persistence of certain operations, then we have to be able to specify that we can override the effects of that constraint. Otherwise, we have to find some way to resolve collisions. To resolve collisions we provide two types of constraints: *operation constraints* and *collision constraints*.

Visualization tasks may attach these two types of constraints to the resources they change. Operation constraints define the scope of the operation effect. First of all, they have a temporal aspect and may either last for the duration of the input device, fade, or be persistent across sessions. Secondly, they may be applied to the entire resource or only to specific values. For example, when doing painting, we may constrain the color parameter of the visualization so that it cannot be used by further operations. Alternatively, we may constrain only the use of the color 'red' so that further painting operations of different colors will not be hindered.

Collision constraints are associated with operation constraints. The collision constraints specify whether the operation constraint that it is associated with can be bypassed. The different conditions in which an operation constraint may be bypassed is as follows:

〈 *Group condition*: The constraint is passed if the group of entities specified in the constraint matches the focus set of the BVI.

〈 *Operation condition*: The constraint is passed if the current visualization task matches the one specified in the collision constraint .

〈 *Undo condition*: The effects of the previous visualization task have to be undone first before the current operation can proceed. If the previous visualization task is a set or chained composite, then the effects of all the basic visualization tasks have to be undone.

6. Previous work

Not surprisingly, much of the work done in characterizing user interfaces [2,5,7] can also be applied to interactive visualizations. This paper builds upon work done in user interface design, in order to come up with a higher-level characterization for interactive visualizations.

In general, not much work has been done in characterizing interactive techniques. Leung [6] came up with a comprehensive framework for understanding various distortion techniques. However it did not address interactive techniques beyond this area.

7. Summary & future work

In summary, we have described a characterization for interactive techniques. In this characterization, we decompose interactive systems into semantic primitives which we call basic visualization interactions (BVIs). We provide a classification for the various primitives and describe how they can be composed to form systems. We have implemented an initial prototype system based on our characterization language. This system allows construction and composition of interactive methods for a diverse set of visualizations. We are working on adding a direct manipulation interface to facilitate assembling and composing these interactive methods.

In the future we will work on expanding our current language with semantics that describe the effectiveness of the various interactive behaviors based on the data, the available graphical objects and the task. This will enable us to provide design knowledge to the user and automatically design dynamic visualizations.

8. Acknowledgments

The authors would like to acknowledge the contributions of John Kolojejchick, Stephan Kerpedjiev, Joe Mattis, and Mark Derthick for comments and insights leading to this paper. This work was supported by a contract from DARPA.

References

1. Chuah, M.C., Roth, S.F., Mattis, J., and Kolojejchick, J. SDM: Malleable information graphics. *Proceedings of the Symposium on Information Visualization*, November 1995, IEEE, 36-42.
2. Foley, J.D., vanDam, A., Feiner, S.K., and Hughes, J.F., *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Company, 1990.
3. Goldstein, J. and Roth, S.F., Using Aggregation and Dynamic Queries for Exploring Large Data Sets, in *Proceedings of CHI'94*, ACM, April 1994, 23-29.
4. Hollands, J. G., Carey, T. T., Matthews M. L., and McCann, C. A., Presenting a Graphical Network: A Comparison of Performance Using Fisheye and Scrolling Views, *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, Elsevier Science B.V., Amsterdam, 1989, 313-320.
5. Jacob, R., A Specification Language for Direct-Manipulation User Interfaces, *Transactions on Graphics*, Vol. 5, No. 4,October 1986, ACM,283-317.
6. Leung, Y.K., and Apperley, M. D., A review and Taxonomy of Distortion-Orientation Presentation Techniques, *Transactions of Computer-Human Interaction*, Vol. 1, No. 2, June 1994, ACM, 126-160.
7. Mackinlay, J., Card, S.K., and Robertson, G.G., A Semantic Analysis of the Design Space of Input Devices, *Human-Computer Interaction*, Vol. 5, 1990, Lawrence Erlbaum Associates, Inc., 145-190.
8. Rao and S. K. Card. The table lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. *Proceedings of CHI '94*, ACM, April 1994, 318-322.
9. Roth, S.F., Kolojejchick J., Mattis J., Goldstein J., Interactive Graphic Design Using Automatic Presentation Knowledge. *Proceedings of CHI'94*, (Boston, April, 1994), April 1994, ACM, 112-117.
10. Roth et al, Visage: A user interface environment for exploring information. *Proceedings of the Symposium on Information Visualization*, IEEE, November 1996.