ABSTRACT
Visualizations are capable of representing many variables and conveying that information efficiently through the perceptual system. Direct manipulation of the visualizations allows similarly efficient operations on the data due to good coordination between the perceptual and motor systems. However, no previous high-dimensional information visualization system has allowed all the displayed data to be edited through direct manipulation. Our design provides powerful visual update propagation rules to avoid manual iteration or script writing when multiple objects must be changed. We found surprising dependencies between the form of the data being edited and the desired update semantics when relations among multiple objects are involved.

Keywords
Direct Manipulation, Information Visualization

INTRODUCTION
Numerous visualization-based interfaces use direct manipulation to update data. For instance, the PalmPilot calendar allows dragging of appointments to change their time or duration. Microsoft Outlook allows adding a category to a task by dragging it within a visualization organized by category. We are concerned with visualization-based environments for exploratory data analysis, including “what-if” scenario development. Our automated design engine, Sage, generates visualizations using graphical properties such as size, shape, and color in addition to spatial properties [1]. So we must design an interface that allows any data to be edited, no matter how it is graphically encoded. Further, these graphics will be used for data and tasks that we cannot anticipate. Editing data will not necessarily be the primary use for the visualization, so we cannot require that the most natural direct manipulation gestures, such as dragging, be used for this purpose. We identified the following design goals:

- Edit any kind of relational or object-oriented data (e.g. continuous, discrete, ordered, nominal, or hierarchical)
- encoded using any visualization technique (e.g. color, text, size, shape, connectivity, indentation, containment, list order, or absolute position).
- Allow updates in the same (analog) modality as the encoding to maintain direct manipulation paradigm for rapid, reversible, and incremental updates.
- Also allow textual updates for precision.
- Do not interfere with other types of direct manipulation (e.g. navigation or brushing), while
- requiring no confusing modalities or time consuming mode switches.
- Edit multiple objects simultaneously, even if non-isomorphic,
- using additive, multiplicative, or identity mapping from changes made to one data object to changes to other data objects.

We use an object-oriented data model, where the objects have attributes and are linked by relations. We call a graphical object that represents a data object a grapheme. Figure 1 illustrates our design. Double-clicking on a grapheme brings up a visualization of all the attribute values that it encodes. For each attribute, there is an iconic handle indicating the graphical property being used for encoding, and a textual representation of the current value. Used as output, the text provides a quick way to see the exact value. It can also be edited to set exact values. The handles are used for analog input. If the data attribute is quantitative, the handle can be dragged to change the value; otherwise mousing on the handle brings up a menu of possible attribute values. In the rest of this paper, we use the word “handle” to refer to both the icons and the text. To avoid confusion between the data and graphical realms, we use “attribute” in regard to data objects, and “property” for graphemes.

Since the input is performed by manipulating dedicated graphical objects, there is no interference with other types of direct manipulation of the grapheme. This is important, since editing underlying data seems in practice to be less common than other operations assigned to direct manipulation gestures in exploratory data visualization systems. Browsing and searching operations are usually the focus, so filtering, scaling, and brushing are given the most direct gestures. As an example consider Visage [2],

Direct Manipulation Input to Data Visualizations

Mark Derthick and Steven F. Roth
Carnegie Mellon University
Pittsburgh, PA USA
mad@cs.cmu.edu, roth@cs.cmu.edu

ABSTRACT
Visualizations are capable of representing many variables and conveying that information efficiently through the perceptual system. Direct manipulation of the visualizations allows similarly efficient operations on the data due to good coordination between the perceptual and motor systems. However, no previous high-dimensional information visualization system has allowed all the displayed data to be edited through direct manipulation. Our design provides powerful visual update propagation rules to avoid manual iteration or script writing when multiple objects must be changed. We found surprising dependencies between the form of the data being edited and the desired update semantics when relations among multiple objects are involved.

Keywords
Direct Manipulation, Information Visualization

INTRODUCTION
Numerous visualization-based interfaces use direct manipulation to update data. For instance, the PalmPilot calendar allows dragging of appointments to change their time or duration. Microsoft Outlook allows adding a category to a task by dragging it within a visualization organized by category. We are concerned with visualization-based environments for exploratory data analysis, including “what-if” scenario development. Our automated design engine, Sage, generates visualizations using graphical properties such as size, shape, and color in addition to spatial properties [1]. So we must design an interface that allows any data to be edited, no matter how it is graphically encoded. Further, these graphics will be used for data and tasks that we cannot anticipate. Editing data will not necessarily be the primary use for the visualization, so we cannot require that the most natural direct manipulation gestures, such as dragging, be used for this purpose. We identified the following design goals:

- Edit any kind of relational or object-oriented data (e.g. continuous, discrete, ordered, nominal, or hierarchical)
- encoded using any visualization technique (e.g. color, text, size, shape, connectivity, indentation, containment, list order, or absolute position).
- Allow updates in the same (analog) modality as the encoding to maintain direct manipulation paradigm for rapid, reversible, and incremental updates.
- Also allow textual updates for precision.
- Do not interfere with other types of direct manipulation (e.g. navigation or brushing), while
- requiring no confusing modalities or time consuming mode switches.
- Edit multiple objects simultaneously, even if non-isomorphic,
- using additive, multiplicative, or identity mapping from changes made to one data object to changes to other data objects.

We use an object-oriented data model, where the objects have attributes and are linked by relations. We call a graphical object that represents a data object a grapheme. Figure 1 illustrates our design. Double-clicking on a grapheme brings up a visualization of all the attribute values that it encodes. For each attribute, there is an iconic handle indicating the graphical property being used for encoding, and a textual representation of the current value. Used as output, the text provides a quick way to see the exact value. It can also be edited to set exact values. The handles are used for analog input. If the data attribute is quantitative, the handle can be dragged to change the value; otherwise mousing on the handle brings up a menu of possible attribute values. In the rest of this paper, we use the word “handle” to refer to both the icons and the text. To avoid confusion between the data and graphical realms, we use “attribute” in regard to data objects, and “property” for graphemes.

Since the input is performed by manipulating dedicated graphical objects, there is no interference with other types of direct manipulation of the grapheme. This is important, since editing underlying data seems in practice to be less common than other operations assigned to direct manipulation gestures in exploratory data visualization systems. Browsing and searching operations are usually the focus, so filtering, scaling, and brushing are given the most direct gestures. As an example consider Visage [2],
the data exploration system being developed by Maya Design Group and Carnegie Mellon that we have implemented the editing interface in. Visage reserves dragging for changing the set of objects shown in a visualization. It uses a second mouse button to offer a menu for navigating to related objects. Single click is used for brushing (selection). Without using modes, it is hard to imagine how editing multiple attributes could be accomplished by direct manipulation of the graphemes. Many systems resort to a property sheet dialog box when editing multiple attributes is required [3]. Our system serves a similar function while retaining more of the original context.

The extra icons are not a bad modality; they are useful visualizations in their own right. Since the handles surround the grapheme, the manipulation is still reasonably direct. Data updates are applied to the objects represented by all selected graphemes.

The next section systematically discusses the choices available for editing various kinds of visualizations and data. Interestingly, some of the problems relate to the data being encoded rather than the graphical method of display, an issue no previous system has addressed. The section “100% Pure Direct Manipulation” argues for the universal availability of editing of all data in all data visualizations. Just as cut and paste allows multiple applications to work together in today’s GUIs, editing together with a small set of other basic operations tailored to data exploration will enable seamless data exploration environments. Finally we discuss future and related work.

DESIGN DECISIONS
Once the basic idea of editing data by manipulating handles representing graphical properties was in place, we still faced a range of possible implementation strategies. Most of the questions come up only for quantitative data attributes; nominal attributes and their menus are simpler.

Controlling Handle Visibility
As a replacement for property sheets, handles for a grapheme should be made visible using the same gesture. In Visage double-click “opens” or shows more detail about a grapheme. This gesture had not previously been used for Sage picture graphemes, so it was an easy decision to rely on it for editing. If a more common operation had usurped this gesture, we would have used a tool palette. Dragging the edit tool to the grapheme would then bring up the handles.

Icon Appearance
The icons should unambiguously represent each graphical property, yet minimize distraction from the visualization. We already used a set of graphical property icons in the SageBrush design interface [1], and the SageBook retrieval interface [4]. For compatibility we used the same icons here. However in populated visualizations the icons introduce an occlusion problem that was not present for the earlier interfaces. We currently make sure that icons are on top of ordinary graphemes, but give them a transparent background. In the figures in this paper, we have added a partially transparent light gray rectangle under the grapheme and handles to help segment them while retaining the context provided by other graphical graphemes. Visage does not provide partial transparency, but this may be a useful technique in other visualization systems.

Non-Spatial Properties
There is little question that attributes encoded as x- (y-) position should be edited by dragging in the x (y) direction. In fact for x and y we use dragging even when the attribute is not quantitative. In this case the visualization itself serves as the menu, and is less disruptive than a pop-up.

For non-spatial graphical properties it is desirable to move in matching spaces as well. However we have chosen to
do all dragging in xy space. Since there is continuous feedback, changing saturation or size without an explicit visualization of color- or size-space is quite natural. We arbitrarily drag in x for these properties. Of course, it is not really dragging, since the grapheme does not move to follow the mouse. We use the word “drag” as a shorthand for holding the mouse button down while moving the mouse.

Non-quantitative attributes encoded non-spatially use menus rather than dragging, so there is no continuous feedback.

**Edit Data, Not Graphic**

Another footnote to our loose use of the word “drag:” Mouse moves are mapped to changes in data values, rather than to graphical changes directly. The graphical properties are then updated to match. This is cleaner because 1) updates due to dragging are handled just the same as, eg, updates from a simulation program, 2) updates first get filtered by the knowledge base. For instance if the attribute ‘price’ is characterized as having a minimum possible value of 0, dragging to the left will eventually halt when this value is reached.

On mouse-down, the mouse position and data value are recorded, and a scale factor is established for mapping mouse displacement to data value deltas. We choose a scale factor so that when the grapheme is repositioned due to the data update, it will find itself at the new mouse position.

**Avoid Hysteresis**

This choice of scaling factors does not always keep the grapheme and mouse in the same position. If the mouse is dragged outside the boundary of the Sage picture, the resulting data value will fall outside the current range of the axis encoder. When this happens, the encoder rescales so that all data can be displayed. We think it important to avoid hysteresis in this case – moving the mouse back to the original position should restore the original value. So we do not change the mouse scaling factor when the encoder rescales. To make a big change in the data value, the user may want to increase the mouse scaling. To do this, she must release the mouse button and grab the handle again, which establishes a new scale.

**OK, Cancel, and Undo**

A drag may be canceled by pushing a second mouse button, and the original data value is restored. Once the mouse button is released, we forget the original value. This avoids the hassle of having an OK button for edits, at the cost of making cancel harder. Visage has a time slider for restoring previous interface states, making undo simple enough that this tradeoff seems warranted.

**Non-Spatial Mouse Scale Factors**

For graphical properties like saturation, we choose a mouse scale factor so that dragging the length of the x-axis changes the data value by an amount equal to the current range of the saturation encoder. This can be thought of as what is done for spatial encoders as well.

**Non-Encoding Properties**

It is often useful to edit the appearance of a visualization independent of any data encoding it uses. For instance a user might want to change the size and color of marks on a chart [that does not use these properties for encoding] to improve legibility and reduce occlusion or to call attention to a subset. The same handles and dragging interface can be used. This ability has not been implemented in Visage, though it was available in a previous version of Sage.

**Derived Attributes**

A visualization may show derived attributes, for example the ratio of the houses’ asking price to their selling price. In this case, reversing the encoding transformation to update the database is ambiguous. If we want to reduce the ratio, do we increase the selling price or reduce the asking price? We regard this as orthogonal to the editing interface design. Current spreadsheets like Excel include “solvers” with sophisticated controls for specifying what can be changed to maximize, minimize, or match a goal value for derived cells. If the underlying application has this ability, the interface can treat setting a goal value for a derived attribute just like setting an ordinary attribute. In Visage, which does not include a solver, most derived attributes are characterized as uneditable. For these attributes the icon and text value are grayed out, and they cannot be dragged or edited.

**Invisible Graphical Properties**

In cases where some attribute values are important while others are not, the visualization may map the latter to a “missing” graphical value. For instance a highway map may have a data type EXIT, with boolean attributes has_service_station, has_hotel, etc. The map will show a hotel icon next to exits with hotels, and no hotel icon otherwise. There are alternative ways to represent this in a database, for instance as a set-valued attribute has_amenities. The Sage picture representation is designed to handle these visualizations, but the Sage designer cannot yet generate them. Hence we have not implemented the ability to edit them either. We imagine the following: In the boolean attribute case, there would be an edit handle for each attribute. Mousing on the icon would bring up a menu containing the values True and False. In the set-values attribute case there would be a single edit handle. Mousing on it would bring up a set of checkboxes, one for each type of amenity. To increase the directness, the Boolean values and checkboxes could be rendered using the domain-specific icons.

**Object Identity Encoding**

So far we have only discussed the case where each grapheme graphically encodes attributes of a single
domain object. Graphemes can also represent tuples of objects related in a given way. Consider the visualization of employees and their secretaries in Figure 2. A mark at \( <x, y> \) means that \( y \)'s secretary is \( x \). Notice that the marks themselves no longer represent a single domain object. In Visage, where all graphemes represent some database object, these marks represent a 2-tuple. We can consider the axes to be labeled based on derived attributes of the tuples: the element_1^\text{name} and the element_2^\text{name}. But we are not really interested in the attribute values for their own sake. It is not helpful to learn that Vincent Gauthier's name is “Vincent Gauthier.” Rather the name is being used to identify Mr. Gauthier, and the useful information being conveyed is that his secretary is John Byrum. So if we were to drag the \( <\text{Gauthier}, \text{Byrum}> \) grapheme in \( x \), we do not want to change Byrum’s name. Rather we want to delete the relation that Gauthier’s secretary is Byrum, and add a new relation such as that Gauthier’s secretary is Heard.

Thus when relations are involved, it becomes necessary to add or remove graphical properties, as well as to change their value. This difference is really in the data realm rather than the graphical one, as similar looking pictures can require very different editing semantics.

The existing \( x \) and \( y \) edit handles can still serve for this delete+add operation. In addition, dropping one of these handles outside the visualization is interpreted as deleting the relation. Using the standard Visage drag modifier to create a copy allows adding a new relation. Figure 3 shows the same data in a network. Here \( x \) and \( y \) jointly encode object identity, so there is a single \( xy \) handle. Dragging it snaps to the nearest object. Somewhat surprisingly, the editing issues here are much more similar to the chart in Figure 2 than Figure 2 is to Figure 1. The data relationships are the key determiner, not the visualization syntax.

**Composed Relations**

Above we assumed that the secretary relationship was specified directly. Now consider that employees are members of labs, and labs have secretaries. Both relations are functional. In the Visage representation, each grapheme now represents an \( \langle \text{employee}, \text{lab}, \text{secretary} \rangle \) tuple, but the lab is not explicitly visualized. What should happen if the \( <\text{Gauthier}, \text{HCI Lab}, \text{Byrum}> \) grapheme is dragged to the Heard column? Either the HCI Lab’s secretary could be changed from Byrum to Heard, or Gauthier could be moved from the HCI Lab to the AI Lab. We treat this case just like derived attributes. There are multiple database states that result in the same visual state. If the application has enough information to resolve the ambiguity, it is free to do so. Otherwise the edit icon should be displayed in gray and not available for editing.

**Multiple Graphemes**

Often the user wants to update an attribute for multiple objects. For instance if one real estate company buys another, he may want to change the \text{listed_by} attribute for all the affected houses. If there is a tax rate increase, he may want to edit the \text{annual_taxes} attribute.

In Visage and many other GUIs, operations such as dragging can be applied to multiple graphemes by using selection. We wanted to use selection to allow editing of multiple objects as well. Several questions quickly surfaced:

**Additive, Multiplicative, and Equality Updates**

In the \text{listed_by} case, the value of all the other selected houses should be set the same as the one being edited, so we call this an equality update. In the \text{annual_taxes} case the percentage change should be the same for all the houses, which we call a multiplicative update. In other cases, the absolute change should be the same, which we call an additive update. Our design offers all three methods, with additive being the default. (Equality is the only choice for non-quantitative attributes.) Each graphical property icon has a +, \( \times \), or = sub-icon. Mousing on the sub-icon offers a menu for changing modes. After every mouse move, depending on mode we determine the difference between the new and original value, the ratio, or just the new value. Objects for the other selected graphemes are then updated. Note that this

![Figure 2](image2.png)

**Figure 2** Chart showing the secretary of each employee.

![Figure 3](image3.png)

**Figure 3** Network showing the same data as Figure 3.
requires remembering the original value for all the objects for the duration of the drag.

**Multiple Symbol Sets**

The previous examples treated visualizations in which all graphemes are isomorphic. Each one uses the same graphical properties to encode the same data attributes, so it is clear how to propagate edits made to one grapheme to the others that are selected. Figure 4 illustrates a more complicated situation. The lower chart adds two marks clustered around each of the bars from Figure 1. We call a cluster of graphemes that represent the same tuple of objects a *symbol*. Each set of isomorphic symbols is called a *symbol set*. Each symbol in the symbol set in the lower chart shows six attributes of its sale:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Property</th>
<th>Grapheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selling Price</td>
<td>y</td>
<td>interval bar</td>
</tr>
<tr>
<td>Date On Market</td>
<td>x1</td>
<td>interval bar</td>
</tr>
<tr>
<td>Date Sold</td>
<td>x2</td>
<td>interval bar</td>
</tr>
<tr>
<td>Lot Size</td>
<td>saturation</td>
<td>interval bar</td>
</tr>
<tr>
<td>Seller Salary</td>
<td>size</td>
<td>left mark</td>
</tr>
<tr>
<td>Buyer Salary</td>
<td>size</td>
<td>right mark</td>
</tr>
</tbody>
</table>

The single-grapheme symbols in the upper chart show the total and average sale price for all sales in Pittsburgh each month:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Property</th>
<th>Grapheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month</td>
<td>x</td>
<td>mark</td>
</tr>
<tr>
<td>Total Sales</td>
<td>y</td>
<td>mark</td>
</tr>
<tr>
<td>Average Price</td>
<td>size</td>
<td>mark</td>
</tr>
</tbody>
</table>

Imagine that we select all the symbols in both symbol sets in Figure 4. Now we drag the *y* property handle of a house sale, updating the sale’s price. For each other selected symbol, we attempt to map the update. For the other sale symbols it is the easy isomorphic case treated above. For the monthly symbols, there are two possible mapping strategies. Either try to match based on the attribute and/or the graphical property. We believe that the feeling of directness is best maintained by mapping based on graphical property. As long as the mapped-to property encodes data of the same type, we propagate the update. Since the *y* property of the monthly symbols encodes Total_Sales, that is what we propagate to.

If we had dragged the *x1* handle instead, there would be no propagation to symbols in the upper chart, because they do not use the *x1* property. If we drag the *size* handle in the upper chart, we would propagate to both the buyer’s and seller’s salary in the lower chart, since both are encoded using *size*.

It is sometimes useful to propagate updates to different graphical properties. By explicitly selecting multiple property icons, edits made to one will propagate to the others. After selecting the *x1* icon of the house sale and

![Figure 4](https://via.placeholder.com/150)
the x icon of a monthly sales, dragging the xl handle would propagate to the date attribute of the monthly sales. This lightweight technique for linking multiple attributes adds power to interactive ‘what-if’ analysis. In a macroeconomic model, a user could maintain a fixed relation between interest rates in several countries for instance. With previous systems, a formula would have to be entered to enable the rates to be updated simultaneously.

A useful side effect of this selection-based strategy is the ability to link multiple properties of a single symbol. For instance by selecting xl and x2 of the house sales, dragging either handle will shift the sale in time while preserving the length of time the house was on the market.

100% PURE DIRECT MANIPULATION

So what have we accomplished? Drag-and-drop of appointments on the PalmPilot is simple and intuitive. Our version requires a double-click to bring up a sprawl of icons and text labels, which are then manipulated rather than the appointment itself. We have sacrificed directness for generality. In this section we argue for the merit of ubiquitous basic operations in an interface. We want every interface to support updating of any data that it visualizes, along with a handful of other universal operations. On the other hand, we do not want overly rigid design guidelines to make our users miserable. If a particular visualization is to be used primarily for input, it should incorporate idiosyncratic extensions to or even override basic operations to make input easy. Thus our editing design should be seen as a backup system that is part of the infrastructure.

Visage embodies this principle of combining universal basic operations and flexible visualizations, called tools, with task specific appliances [5]. The editing interface is a tool, while the PalmPilot calendar is an appliance. Just as tasks are accomplished within a larger context, appliances should be smoothly integrated into a larger environment. Effective environments will provide user interface mechanisms that support multiple information analysis tasks, including

- searching for, examining, and narrowing the scope of information to relevant subsets
- controlling the level of detail of data
- reorganizing, grouping and transforming data to create new information
- computing new attributes derived from others
- detecting important relationships and patterns
- communicating the results of analyses to other people
- acting on this information through computer tools for executing orders
- editing the information

Our design goal has been that of creating concrete, external, manipulable objects to enable people to perform tasks whose abstractness, complexity, or magnitude make them very difficult. This includes understanding how to create a consistent information workspace where people can make seamless transitions in their use of multiple visualizations, tools, and applications. Visage is our attempt to provide these abilities, through a relatively complete set of basic operations upon which every application implemented within it can draw, including

- drag-and-drop objects from one frame to another
- navigate (drill-down) from object along a relation
- brush (select) an object or a set of objects in a choice of colors
- aggregate (roll-up) a set of selected objects to create a new object with properties computed from its members
- copy an object or a frame
- dynamic query
- create a new frame either from a palette of standard frames or with Sage

The specific operations may differ in other systems, but the result of the operations should depend only on the underlying data object, and be uniformly applicable in any part of the environment. An interface with this regularity is called information-centric [2]. It leads to predictable behavior even in unfamiliar specialized frames.

The information centric approach is the natural evolution of direct manipulation. When first proposed, direct manipulation stood in contrast to command line interfaces [6]. The difference was primarily that between commands and manipulation. As manipulation of visual objects has ubiquitously replaced command processors, the power of the interfaces has also increased. This results in hundreds or thousands of nested menus, dialog boxes, and options. The experience in these cases is not so direct; the visual input widgets no longer seem to represent anything meaningful. The direct manipulation criterion that input=output is often lost.

The information-centric paradigm, more restrictive than today’s GUI design guidelines, most fully embodies the desirable characteristics of direct manipulation – incremental, reversible, continuous feedback, enhanced expert performance, memorability, and increased control [6]. Therefore all interfaces should be visualizations and vice versa. Mentioning similar concerns about today’s GUIs, Frohlich calls attention to visualizations and terms the subset of operations they support “pure direct manipulation” [7]. Rao uses the phrase “see and go” [8].

Our goal is to make all interfaces see and go. The editing technique is one step on a long road to a data exploration environment with a comprehensive and consistent
“physics.” Point solutions will never add up to a seamless environment where users can attend to their data and tasks rather than being aware of multiple tools.

FUTURE WORK
The primary advantage we see for our editing interface over previous [indirect] input methods for general purpose data visualizations is the increased immediacy. The user can stay focused on the visualization, and feedback is continuous. To test the method we want to identify users who do much data editing in pursuit of problem solving, and who preferably already use visualizations as an aid. During Fall semester 1999, a group of HCI masters students at CMU will be conducting studies of financial analysts who use complicated spreadsheet models to solve business problems. This will include observation of current practice, and lab studies of tools enhanced with the ability to edit data through direct manipulation. It will be interesting to learn the extent to which analysts find visualizations useful as output in the base case, and whether they find them more useful when both input and output can occur in visualizations.

RELATED WORK
Designing editing interfaces in a general way has not been an issue for hand-built visualization applications. We are not aware of any that use anything other than dragging, changing grapheme size, or typing text. For GUI builders and automated visualization design, a general framework is required. A number of constraint-based systems allow graphical encoding of data by connecting application variables to handles on graphemes \[9, 10\]. Handles are typically provided for the spatial properties \(x\), \(y\), \(w\), and \(h\). Lapidary \[3\] includes a “new point” interactor handle as well. Sometimes these handles are intended for use only at design time, while in other cases they are available for end-user input as well. Due to the large number of properties that Sage uses for encoding, our handles present more of an occlusion problem than previous systems. For instance, Gilt unobtrusively uses 12 handles for editing spatial properties:

To edit non-spatial properties, most previous systems use a property sheet dialog box \[3\]. This abandons the contextual environment of the visualization, usually allows only text or menu input rather than analog, and uses an OK button rather than giving continuous feedback. On the other hand, having a separate window avoids the occlusion problem our handles have. It could be a useful option in our design to drag a copy of a grapheme to a separate edit tool window. The appearance of the handles would remain unchanged – still affording analog updates with continuous feedback.

Although not a data visualization system, Adobe Photoshop uses this approach for many of its pop-up windows. Sliders are available for setting parameters such as transparency, and continuous feedback is provided.

EUPHORIA \[10\] is the GUI builder and User Interface Management System that has paid the most attention to editing data using direct manipulation. It allows user design of visual interfaces by connecting widget handles to data variables and imposing constraints. The same handles can be used as direct manipulation inputs to edit the data. Thus any data that can be visualized can also be edited. This is very similar in spirit to our interface. All current widget handles are for encoding data spatially. The designers planned to include handles for color, and to use a property editor for other non-spatial properties \[T. Paul McCartney, personal communication 1999\]. The issues of hysteresis, multiple symbols (not to mention multiple symbol sets), and object identity are not addressed.

No previous system allows simultaneous editing of attributes encoded with different graphical properties, as we can do by selecting multiple handle icons.

GUI builder’s require less understanding of the underlying data than automated visualization design systems. Yet even here no previous system has made the distinction between attributes displayed for their own sake and those used for object identification. BOZ \[11\] perhaps comes closest. It designs visualizations appropriate for a given user task and data. Casner drops tantalizing hints that these tasks can include data updates, mentioning that there are TELL as well as ASK, logical operators. But no rules for realizing TELL operators as interface gestures are given.

Spreadsheets are an example of constraint systems based on a clear model of how the underlying data is manipulated. A method for fluid spreadsheet visualization \[12\] shows formulas explicitly and allows direct manipulation update to them. Cell references can be absolute or relative. In some ways this is analogous to our +, \(\times\), and = modes in that it does not affect the original formula (grapheme), but only copies (other selected graphemes).

CONCLUSION
This paper has described an extension to the kind of widget handles provided by GUI interface builders, adapted to data visualizations. This required adding handles for non-spatial properties, manipulating multiple selected graphemes in +, \(\times\), or = modes, and disambiguation based on whether object identity is involved. Using selection to link arbitrary grapheme properties so that updates apply to all appears quite powerful.
The visualization description language previously developed for the Sage project led us to a principled treatment of issues that previous systems have not addressed. The vocabulary of symbol sets, symbols, graphemes, and graphene properties provides the means for the sophisticated linking of multiple updates. The notion of object-identifying attributes distinguishes between two semantic interpretations for the same editing gesture – changing an attribute versus changing a relation.

In regard to the design goals listed in the Introduction, we can edit all data displayed in all visualizations that work by encoding data attributes as graphical properties. We believe this includes all visualizations of relational or object-oriented data. The editing can be analog with continuous feedback, or using text for precision. Direct manipulation operations on the graphemes are not impaired, and there is great flexibility to edit multiple objects simultaneously.

The main drawbacks are that manipulation of handles is less direct than operations on the graphemes themselves, and that the handles occlude the visualization. We argued that this admittedly cumbersome interface is an important backup providing universal editing ability for any visualized data. As a backup, it should be compared with the even more cumbersome and less direct dialog boxes and property sheets of other systems, rather than with point solutions to direct manipulation data updates. Universal basic operations enhance the separation between application data and visual interface, providing greater flexibility and a consistent “see and go” model to the end user.

REFERENCES


