Chapter II:
Visualization Techniques Framework

A Functional Description

The goal of this thesis is to integrate data processing decisions into the design process of an automatic visualization system. This work enhances the quality and breadth of visualization designs that can be automatically generated as well as expands the range of tasks that can be addressed by an automatic system. In order to integrate data processing operations into automatic design, we must first understand what data processing operations are available, how they can be applied to data elements within a visualization, and how they may be combined together with data-to-graphical mapping operations[1]. To achieve this, we analyze existing visualization systems, and develop a framework or layer of abstraction for understanding current visualization techniques, the types of functions they are composed of (including data and mapping functions), as well as how they are built, combined, and used.

Creating this framework, however, is a difficult task. The widespread development of new visualization techniques in recent years, due to significant increases in information processing demands, have left them fragmented, making it difficult and expensive to combine, customize, or generalize their functionality. Visualization systems are often written for a variety of domains and exist at many different levels of granularity. In addition, they provide a wide range of functions that operate on such disparate objects as inputs devices (*scroll-list, bounding-box*), data concepts (*houses, people*), data attributes (*selling_price, num_rooms*), graphical objects (*marks, interval-bars*) and graphical properties (*color, shape*). Techniques that appear to be physically identical may share very little functional similarity and vice versa.

In this chapter and the next we present a visualization framework that models the functional operations (including data and mapping operations) within various visualization techniques as well as the relationship of these

[1] Mapping operations capture how data elements can be mapped to graphical elements, so that complex cognitive processing tasks can be offloaded onto our perceptual system. Previous automatic systems only considered the use of mapping operations. This thesis expands automatic design to include both mapping and data functions.

functions to other visualization elements such as input-devices, data concepts and graphical objects. This framework allows us to effectively create and customize visualization techniques as well as enables us to integrate a large set of powerful functions into our automatic visualization design system, thereby increasing its communicative and design effectiveness. Note that rather than only capturing data and mapping operations, as is needed by our automatic designer, we decided to establish a wider framework that covers all visualization design functions (i.e. including data, mapping, graphical and rendering functions). This broader framework is flexible, and provides us with a better understanding of the role that data processing transforms may play in design, not only with mapping functions, but also with graphical and rendering operations. This broader framework is also easily extensible so that graphical and rendering visualization designs can be integrated into automatic systems in the future.

Our visualization techniques framework stands as a contribution of its own and can be used to characterize and capture the state of visualization techniques today. Contributions of our framework include:

1. *Visualization function primitives*: Our framework presents a set of primitive functions that commonly occur in visualization techniques. These primitives form the basic building blocks of our automatic design system. In addition they give us a better understanding of the class of tasks that can be achieved by different visualization methods and allow us to consider such methods at the same level of granularity (by decomposing them down into the same set of primitives). We show in section II-3 that this set of primitives can express a wide range of visualization methods. These functional primitives can also provide the basis for establishing a visualization techniques library which will simplify the process of creating interactive visualization systems.

2. *Composition rules for merging visualization primitives and exploring the design space*: Once we have defined a set of primitives, we specify rules that determine how these primitives can be combined to form more complex behaviors. These composition rules are very powerful because they allow us to generate an infinite set of visualization techniques from a small set of primitives. By combining together components of existing visualization methods, we can adapt these methods to serve in new domains, devise interesting new ways of achieving tasks, and begin exploring and expanding the design space of visualization techniques.

3. *Visualization independent specification of visualization techniques*: Our framework provides a general language for specifying visualization techniques that is not tied to any particular visual representation. Once specified, a visualization technique may be easily attached to a variety of visualization designs. Such flexibility increases the effectiveness with which we are able to generate, prototype and test visualization techniques.

4. *New design methodology for analyzing the interactive design space*: Our framework presents a two-level design methodology for creating visualization techniques: the *functional level* and the *instantiation level*. The functional level is a more abstract level of characterization that allows us to group, categorize, and reason about techniques based on their functionality and application to tasks. The instantiation level, on the other hand, characterizes techniques based on a set of low-level primitives. At this level we capture all the specifics within a

technique so that based on the instantiation description we can fully generate a visualization interface. Because the instantiation description is detailed and low-level, it is difficult to make generalizations about the various visualization techniques, unlike in the functional level. This low-level description, however, is necessary for our automatic design system because it must be able to describe and generate instantiable or realizable designs. In addition, primitives at the instantiation level form a useful visualization API set. Our framework describes both the functional and instantiation levels as well as presents a systematic process of how to move from a functional description into an instantiable description. This is an advance over previous frameworks that only considered either one of these levels in isolation. Refer to appendix B-1 for a more complete discussion of the differences between our framework and previous work.

In this chapter we describe the functional level of our design methodology. In particular we show how visualization techniques can be functionally decomposed into two primary components, *object definition* and *transformation*, as well as how these two-component techniques can be combined to create interesting behaviors. Our automatic design system later uses this object definition/transformation framework (*ODT framework*) to build and generate visualization designs that utilize both data processing and mapping functions. To illustrate the generality and applicability of our framework, we will also show how it can be used to map out part of the visualization techniques design space, and give some interesting observations made from analyzing that space. Readers who are only interested in the automatic design aspects of this thesis can skip section II-3 of this chapter as it pertains to the generality and scope of the framework rather than to its use in our automatic system. By using our framework to explore current and future techniques, however, we show that it is not only useful to our central thesis in automatic design but also generally applicable to the analysis of a large range of techniques (some of which are not currently captured by our automatic design system). The next chapter will explore visualization design at the concrete instantiation level, as well as evaluate the entire framework based on completeness, coverage, and practicality.

II-1 Visualizations & Visualization Techniques

A *visualization* is a graphical rendering of a set of data attributes. The process of creating a visualization begins with the data transformation phase as is shown in Figure II-1. *Data transforms* are used to calculate derived results or summarize attribute values within a data set. For example we can use a *subtraction* data transform to compute the duration that a house stays on the market from the *date_on_market* and *date_sold* house attributes. Alternatively we can use the *mean* data transform to summarize the selling price of all houses in the *Shadyside* area. Data transforms can also be used to compute new meta-data from an existing data set such as the number of times a particular object appears, or the alphabetical or numerical ordering of a set of values. Data transforms exist solely in the data realm and are used to generate new data concepts and values based on the existing data set.

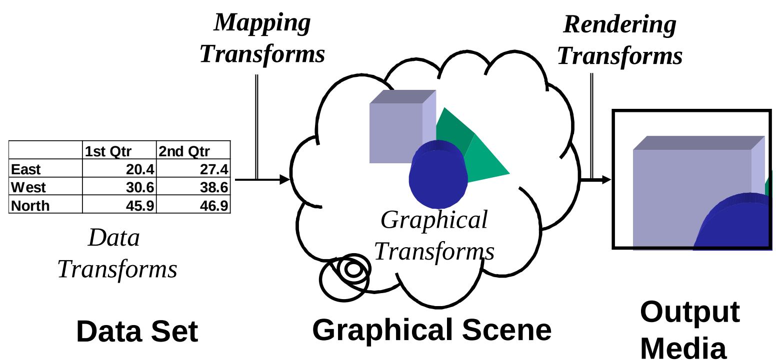
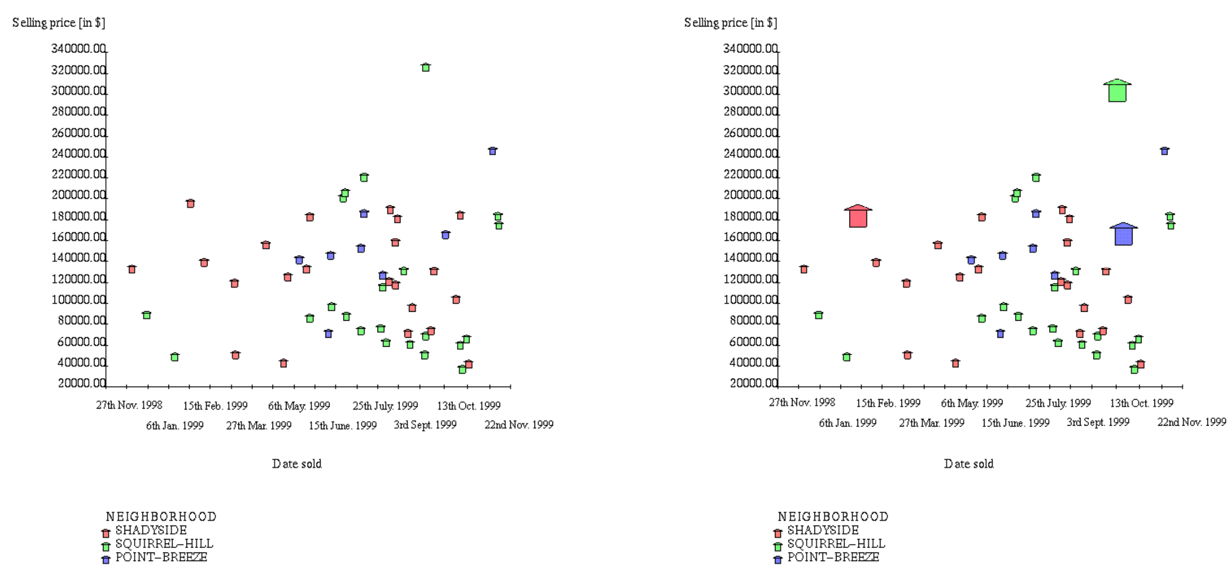


Figure II-1: Visualization generation process consisting of four transformation classes (data, mapping, graphical, and rendering) across three different realms (data, graphical, output media)

Once we have processed all the necessary data, we proceed to the mapping phase where data concepts are mapped to graphical objects and data attributes are mapped onto graphical properties. For example, the visualization in Figure II-2a contains four mapping transforms. An object mapping transform represents all house concepts with graphical *marks*, and a set of attribute mappings link different data attributes of the house data concepts such as *selling_price*, *neighborhood* and *date_sold* to different graphical properties of the *mark* graphical objects such as *x-position*, *color*, and *y-position* respectively. At the end of the mapping phase we would have constructed a *graphical scene* representing the data we want to show in the visualization. A *graphical scene* is an abstract model of a boundless space, capturing the position, relationships and appearance of all visual objects (e.g. *marks, axes, legend, labels*) within a visualization.



(a) Several mapping transforms are used to show the *house* data concepts and some of its attributes

(b) Same visualization as Figure II-2 but several house objects have been enlarged with a graphical transform

Figure II-2: Example visualization with house data. Each mark represents a *house* data concept. The *x-axis* shows *date_sold*; the *y-axis* shows *selling_price*; and *color* shows *neighborhood*.

Graphical transforms are used to change the appearance of objects within a graphical scene. For example in Figure II-2b graphical transforms are applied to several selected or focus graphical objects from Figure II-2a so that they appear larger and more salient than the other objects in the visualization. Note that these *size* enlargements do not correspond to any information in the data set and thus cannot be appropriately shown with mapping transforms. Graphical transforms can also be used to change other graphical properties (e.g. *color*, *shape*, *position*) and other graphical object classes such as *legends*, *axes* or even the entire *chart region*.

Finally in the rendering process, abstract graphical objects in the graphical scene are transferred onto a bounded output media such as paper, a physical 3D model, or a CRT screen. Figure II-2 shows a rendering of a visualization design on paper. Different media types constrain the classes of techniques that can be used as well as their effectiveness. In this thesis we will only consider the use of CRT screens, thus in our work rendering transforms describe for each screen pixel, the part of the graphical scene to which it corresponds. Note that rendering transforms are the only transform class that operates on physical screen space. All other transformation classes operate on abstract objects such as data concepts and data attributes or graphical objects and graphical properties. The visualization generation process presented here is based on previous work by Card et al.[*Card*, *1999*] and Chuah & Roth [*Chuah*, *1996*]. These four transform classes in the visualization generation process (Figure II-1) form the basis of visualization techniques.

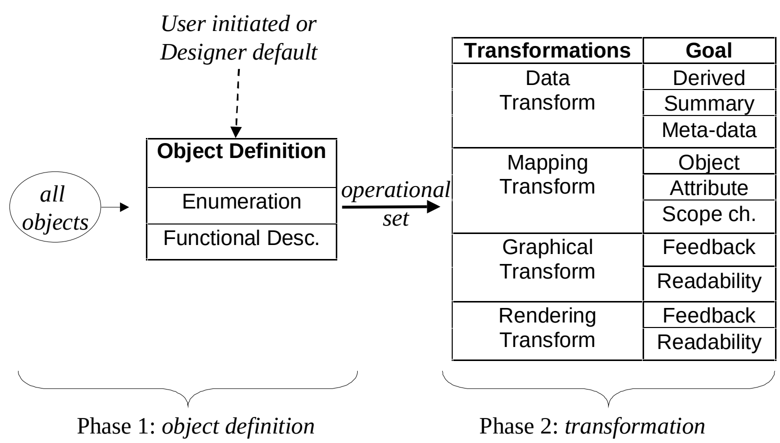


Figure II-3: A visualization technique is defined in this work to contain two components (object definition and transformation). Object definition can be achieved through enumeration or functional description. Transformation can be achieved through data, mapping, graphical, or rendering functions. The transformation functions can be further divided based on their goals.

In this thesis, a *visualization technique* is defined as having two components, an *object definition* component and a *transformation* component (Figure II-3). In the *object definition* component, we define a set of elements that can be from any of the three realms (*data*, *graphical* or *output media*) in Figure II-1. For example, an interaction may operate over a set of *house* records, a set of graphical *marks* and *bars*, or even a display space within the visualization window. The resulting set of elements from the object definition component (*operational set*) is subsequently processed in the *transformation* component according to our current goals. These transform functions can be used in a multitude of ways to solve different tasks, thus, apart from specifying the transformation class it is also useful to capture the general goal(s) or effects of these visualization function primitives. The goals of data and mapping transforms, which occur at the start of the visualization generation process, are usually to prepare and set the contents of a visualization. The goals of graphical and rendering transforms, which occur at the end of the visualization generation process, are to provide users with feedback on a visualization technique, or to enhance the readability of the visualization content defined in the data and mapping transform stages.

Visualization techniques allow us to create new visualization designs or modify existing ones. A set of mapping transforms and at least one rendering transform are requisites for creating a new visualization. These transforms are necessary to convert a set of data values to a rendered visual representation of that data. On the other hand data transforms are only necessary if the data set is not in the desired state to begin with and must be further processed. Graphical transforms are needed only when there are readability, feedback or rhetorical requirements. While data and graphical transforms operate within a single realm, mapping and rendering transforms operate across realms, expressing relationships between different object classes. Mapping transforms relate data concepts to graphical objects and rendering transforms relate the graphical objects within a graphical scene to an output media such as the CRT screen. Apart from data, graphical, or media objects, a visualization technique may also be attached to input-devices that allow end-users to interactively alter a technique's functionality or results even if only in a limited way. Techniques that are attached to input-devices are commonly referred to as *interactive techniques*. Subsequently we describe the two visualization technique components: object definition and transformation.

II-1.1 Object Definition Component

(a) Before any data attribute constraints are set

(b) After data attribute constraints are set

Figure II-4: Dynamic Query Sliders applied to house data. Each *bar* encodes a *house data concept*; *x-axis* encodes *date_on_market* and *date_sold*; *y-axis* encodes *house_address*. There are two dynamic query sliders [Ahlberg, 1992], one allows users to place constraints on the *num_rooms* data attribute and the other allows users to place constraints on the *selling_price* data attribute. Houses that do not fulfill constraints become non-visible as in (b).

The object definition component may be initiated by a user or it may be preset as a system default by the designer of the technique. For example, the dynamic query slider technique [*Ahlberg*, *1992*] allow users to search for data elements by placing constraints on their attributes. Constraints are placed by setting data attribute threshold values through the use of input-devices such as *sliders* (e.g. as in Figure II-4). Once the constraints are set, only those data concepts that fulfill the search constraints are shown. Thus the dynamic query slider technique lets users manually initiate object definition by controlling a *slider*.

The visualization system shown in Figure II-5, on the other hand, uses system default objects. Figure II-5 shows a campus map of Carnegie Mellon University with a "Next Lot" button. Pressing this button will cause one campus parking lot to be shown in red. Pressing the button again will cause a different campus parking lot to get highlighted and so on. The particular parking lot to highlight is preset by the system designer. Morewood Parking gets highlighted first, followed by the Parking Garage, etc.

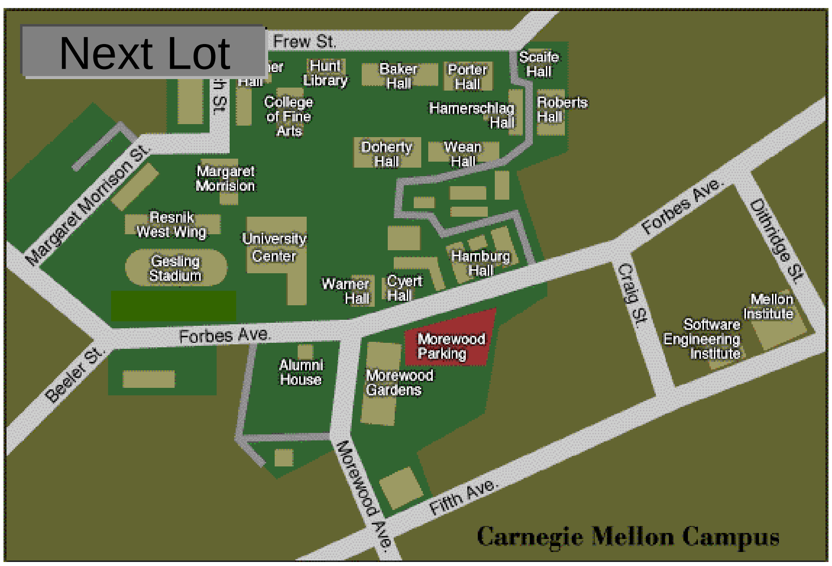


Figure II-5: Map showing the different parking lots at CMU (borrowed from http://www.cmu.edu). Clicking the *next-lot* button will cause a predefined parking lot to get highlighted red (e.g. *morewood parking*). Subsequent presses to the *next-lot* button will cause subsequent parking lots to get highlighted.

There are two primary object definition methods: 1) enumeration or 2) functional description as is shown in Figure II-3. The highlighting technique used in Figure II-5 performs object definition through enumeration because the designer of the graphic explicitly named each and every car park lot on the map. The dynamic query slider technique on the other hand, defines the operational set through functional description. I.e. the object set is captured through a mathematical function applied to object attributes rather than by explicit naming. For this technique, the function used is a simple *greater-than* or *less-than* threshold operator.

II-1.2 Transformation Component

After the operational set has been defined, we manipulate and modify it in the transformation component. There are four classes of transformations, corresponding to the four phases of the visualization generation process: 1) data transforms, 2) mapping transforms, 3) graphical transforms, and 4) rendering transforms. Previous automatic design systems only considered the use of mapping transforms. In this thesis we expand the automatic design process to consider data processing transforms as well. However we leave consideration of graphical and rendering transforms in automatic design for future work. By capturing all four transformation classes in our framework, however, we can more clearly and completely see the roles that data and mapping functions can play in visualization design and in solving data analysis problems.

<u>*1. Data Transforms*</u>

There are an infinite number of ways with which we can process the information within a data set. As a result a data set may not always contain information that is of interest to us in a form that we desire. By using data visualization techniques we may direct the system to generate derived data attributes that fit our task requirements. Data visualization techniques can commonly be found in spreadsheet programs and data analysis software where a set of different mathematical computations can be applied to selected values in the interface.

<u>*2. Mapping Transforms*</u>

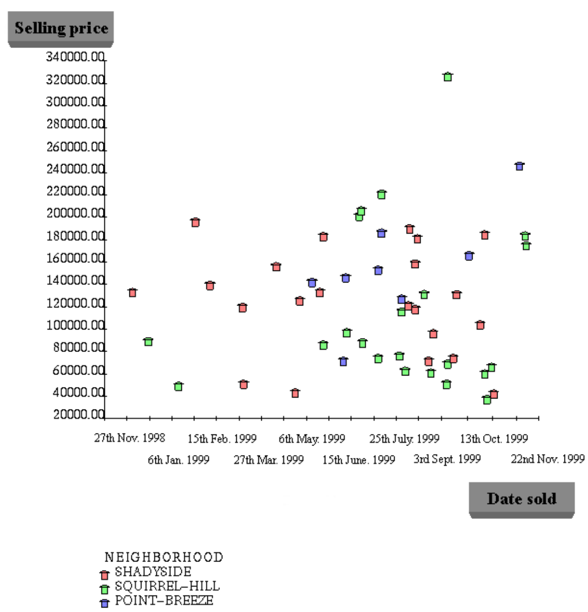


Figure II-6: Visualization system that allows re-mapping of data attributes to the two positional axes. Each *mark* in the visualization represents a *house* data concept. Currently the *x-axis* is set to encode *date_sold* and the *y-axis* is set of encode *selling_price.*

Mapping transforms are most commonly used to encode data concepts and data attributes with graphical objects and graphical properties. To create a new visualization, one or more mapping visualization techniques are required. Mapping techniques commonly operate on all the data concepts in the data set and the input transformation parameters (e.g. *selling-price*, *neighborhood*, *date-sold* and *x-position*, *color*, *y-position*) are predefined by the visualization designer. I.e. object definition is achieved through designer default functional description. It is however not necessary for all the mapping transform arguments to be predefined. For example the visualization in Figure II-6 has option menus attached to both the *x-axis* and *y-axis* of the visualization so that users may pick different attributes to re-map to the two axes. Apart from mapping data to graphics, mapping transforms can also be used to change the scope of existing data-to-graphical encodings. For example we could apply the object and attribute mappings in Figure II-6 to only the *pink* objects by using a *scope* mapping transform.

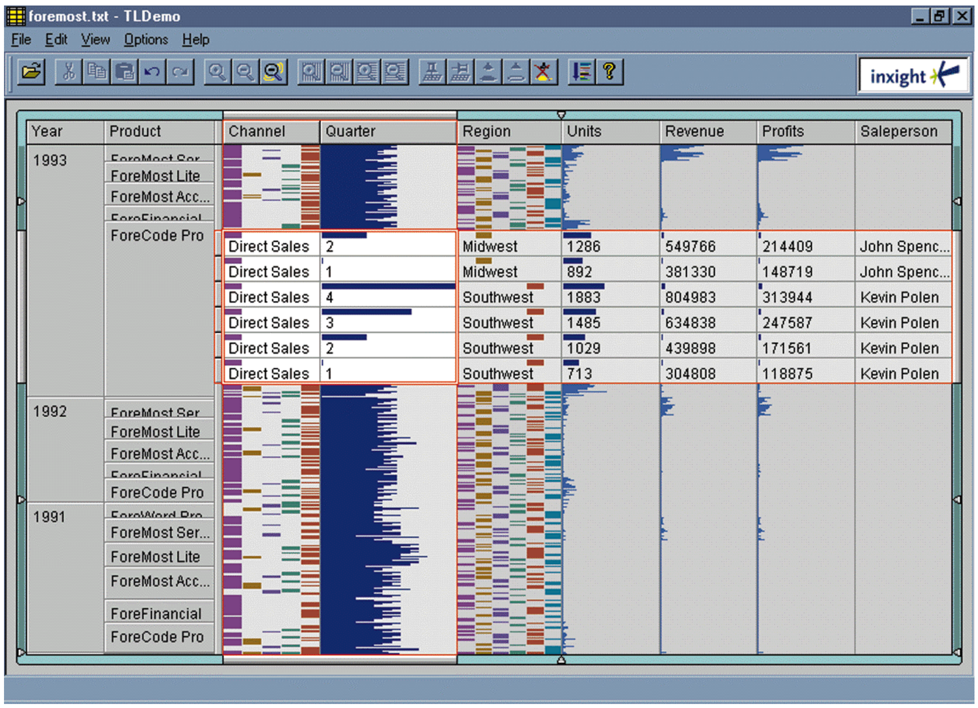
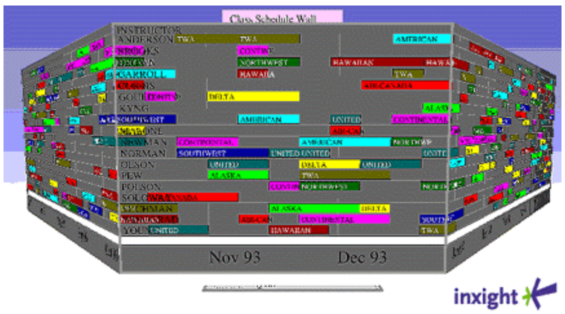


Figure II-7: TableLens System [*Rao, 1994*] (borrowed from www.inxight.com)

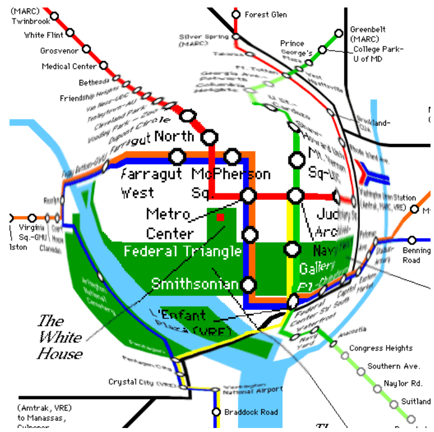
Mapping transforms can also be used in more elaborate ways. The PAD++ [*Bederson, 1994*] and the TableLens [*Rao*, *1994*] systems use mapping transforms to achieve "semantic zooming". In these systems, the graphical representation classes (e.g. *mark*, *text*, *bar*) used in the mapping transforms change based on available screen space. When more screen space is available a more accurate graphical representation is used to show the data and when less space is available a visually simpler but less accurate graphical representation is used instead. For example, the TableLens system (shown in Figure II-7) uses *bars* to represent the data concepts when less space is available and *text* (in addition to *bars*) when the space gets magnified.

3. Graphical Transforms

Graphical transforms are commonly used to provide feedback or to increase the readability of a visualization. The most common use of graphical transforms is for manipulating un-mapped graphical property values to improve the readability of a visual design (e.g. layout operations) or for providing simple feedback that reflect state changes (e.g. feedback indicating that a set of objects have been selected). Graphical transforms may also be used as a rhetorical device, e.g. enlarging the objects currently under discussion. Note, however, that graphical transforms are not limited to un-mapped graphical property values. When used to change mapped graphical properties, however, they may distort interpretation of the data set as was discussed in Chuah et al. [*Chuah*, *1995*], thus such techniques should only be used with extreme caution and adequate user feedback.



Perspective Wall
(borrowed from Xerox PARC
User Interface Research Group page,
http://www.parc.xerox.com/istl/projects/uir/projects/InformationVisualization.html)



Fisheye non-linear image magnification
(borrowed from
http://www.cs.indiana.edu/~tkeahey/research/fad/fad.html)

Figure II-8: Example rendering visualization techniques

4. Rendering Transforms

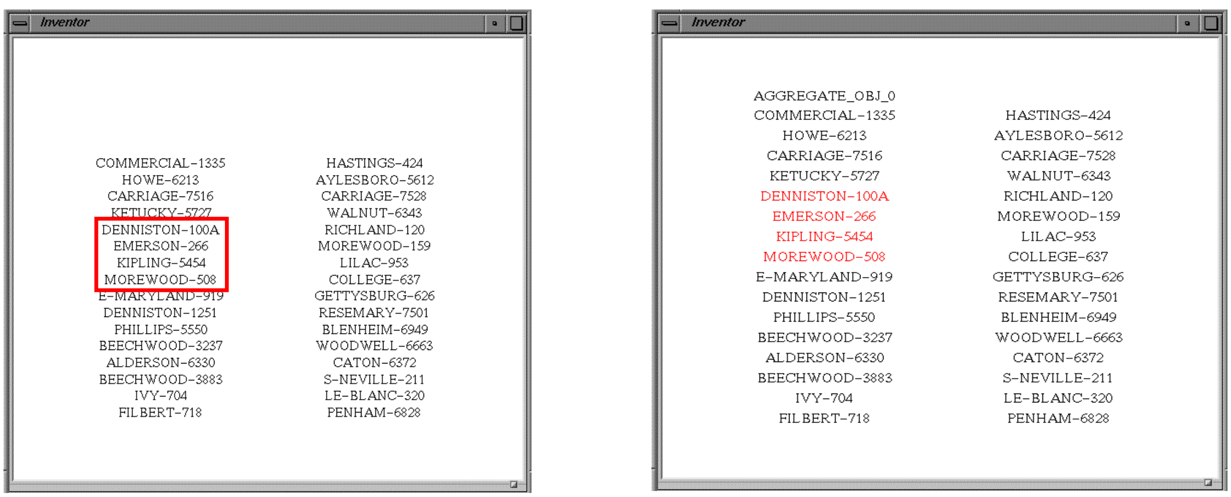
Like graphical techniques, rendering techniques are also used to give users feedback or to improve the readability of a visualization. Many interactive systems today use rendering transforms in interesting ways to distort the graphical scene so that users may focus on particular parts of the scene while maintaining context of the surrounding areas. Some examples include the Perspective Wall [*Mackinlay, 1991*] (shown in Figure II-8a), Fisheye Lens distortion [*Furnas, 1991*] (shown in Figure II-8a), and Table Lenses (shown in Figure II-7). Such techniques improve the readability of visualizations by reducing object occlusion and output density around the focus objects so that their visibility is increased. Apart from distortion techniques, rendering transforms are also used to achieve more common navigation techniques such as *zoom* and *pan*. Appendix F discusses rendering transforms in greater detail and show how they can be used to solve readability problems.

In summary, a primitive visualization technique consists of two components: an object definition component and a transformation component (shown in Figure II-3). The object definition component may be initiated by the user or be set as a system default. There are two ways in which objects may be defined, either through enumeration or functional description. Once defined, the selected visualization elements may be transformed using data, mapping, graphical, or rendering functions. These general transform classes can further be categorized based on their goals or effects.

We will build upon this object definition/transformation (*ODT*) model in this chapter and the next. In chapter V we show how we apply this *ODT model* in our prototype automatic design system and how this model allows us to create visualization designs that contain both data and mapping transform functions. The *ODT model*, however, is useful beyond our automatic design system because it enhances our ability to create, and customize visualization techniques, as well as enables us to explore and organize the visualization techniques design space (section II-3).

II-2 Composition

In the previous section we presented a simple ODT model which decomposes a visualization technique into a single object definition operator and a single transformation operator. Many common visualization techniques, however, are more complex and may combine multiple object definition and transformation functions. For example, consider a simple aggregation tool. Aggregation or binning is commonly used for dealing with large data sets. When there are many data concepts, it is difficult to assimilate and analyze all of the data simultaneously. To reduce the data set to a more manageable size, we combine multiple data concepts together and represent them through a single "aggregate" object. There are a wide variety of aggregation methods [*Goldstein, 1994; Chuah, 1998*]. Here we consider a simple aggregation technique that lets users select a set of objects from a visualization display through a *bounding-box*, and then creates a new aggregate object (summary object) containing the selected set. This aggregation process is shown in Figure II-9. First the user selects *denniston-100A, emerson-266, kipling-5454* and *morewood-508* using a *bounding-box* (Figure II-9a). These objects are then grouped to form an aggregate object (*aggregate-object-0*) that appears at the top of Figure II-9b.



(a) Select the set of objects to aggregate

(b) *Aggregate_obj_0,* is generated. *Aggregate_obj_0* contains objects *denniston-100A, emerson-266, kipling-5454* and *morewood-508* that are highlighted in *red*

Figure II-9: Simple aggregation technique. *Text* encodes *house_address*. By using this technique users get to select a set of *house* data concepts using a *bounding-box* and aggregate or group them together to form an aggregate object (e.g. *aggregate_obj_0*). Unlike techniques in the previous section, this aggregation method utilizes multiple transformation methods including a graphical transform to highlight the selected objects and a data transform to summarize the underlying data concepts of the selection.

Object definition through a *bounding-box* falls into the user-enumeration object definition category. Once selected the set of data concepts are summarized using the *group-objects* data transform that maps a set of data concepts to one representative group object as is shown above. Note that in addition to the summarization (*group-data*) data transform, it is useful to give users some feedback as to which objects are being summarized. This is achieved by adding a graphical transform to highlight the selected objects, in addition to the summary data transform. In Figure II-9b, for example, the aggregated objects are highlighted *red*. Such multi-transform techniques are made possible through composition operators that allow us to combine and generate many rich and interesting interactive behaviors from simple two-operator techniques.

There are four main classes of composition: 1) object definition composition, 2) transformation composition 3) producer-consumer composition, and 4) partition composition.

II-2.1 Object Definition Composition

Object definition composition is used when we want to apply the same transformation methods to combinations of multiple object definition sets. Object sets are combined using *set-operator* functions. Set operator functions such as *union*, *difference* and *intersection* take in one or more object sets as input and produce a single output set based on the membership of the input sets.

For example consider the multiple-constraint dynamic query slider technique shown in Figure II-4. This interface has two sliders, one constrains the *selling_price* attribute and the other the *num_rooms* attribute. These two primitive techniques can be expressed using the object definition/transformation model (*ODT model*) as is shown in Figure II-10a. For a complete description of the diagrammatic notations used in these *ODT diagrams* refer to appendix A-1.

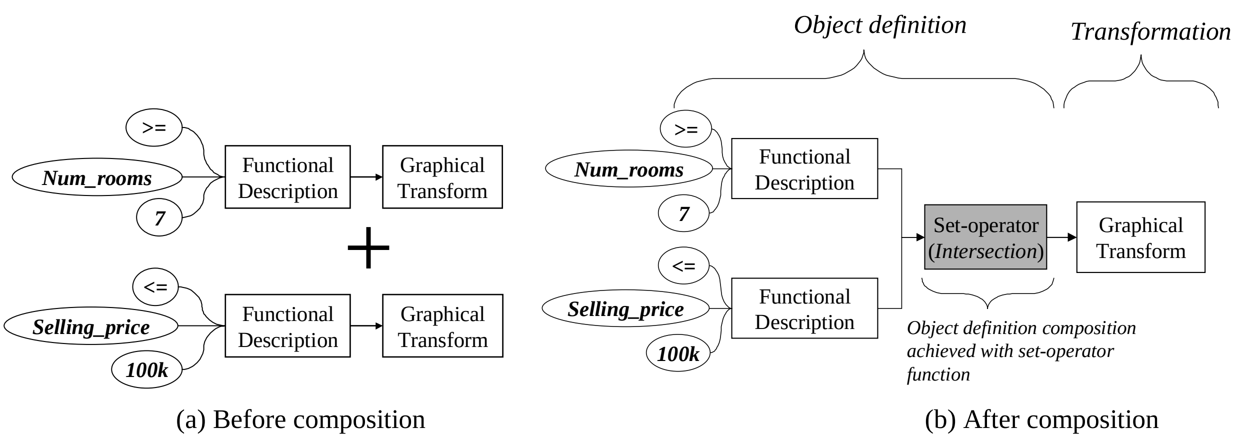


Figure II-10: Object definition composition for the multiple constraint dynamic query technique shown in Figure II-4. The diagrammatic conventions and notations used in the specifications in this chapter and the next are described in appendix A-1.

Expressing the two constraint techniques separately as in Figure II-10a, does not capture the relationship between them that requires both *selling_price* and *num_rooms* constraints to be fulfilled simultaneously. To express this relationship we must perform object definition composition and apply an *intersection* set-operator function to combine the two constraint sets as in Figure II-10b. This will cause only those objects that pass both constraints to be graphically transformed. Set-operator functions are used to further refine selected object sets, thus they are considered part of the object definition component.

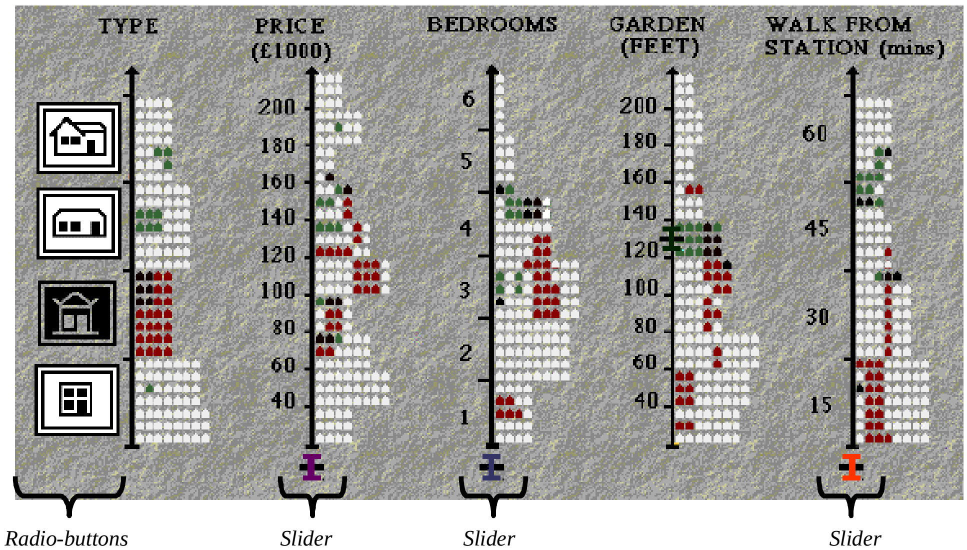
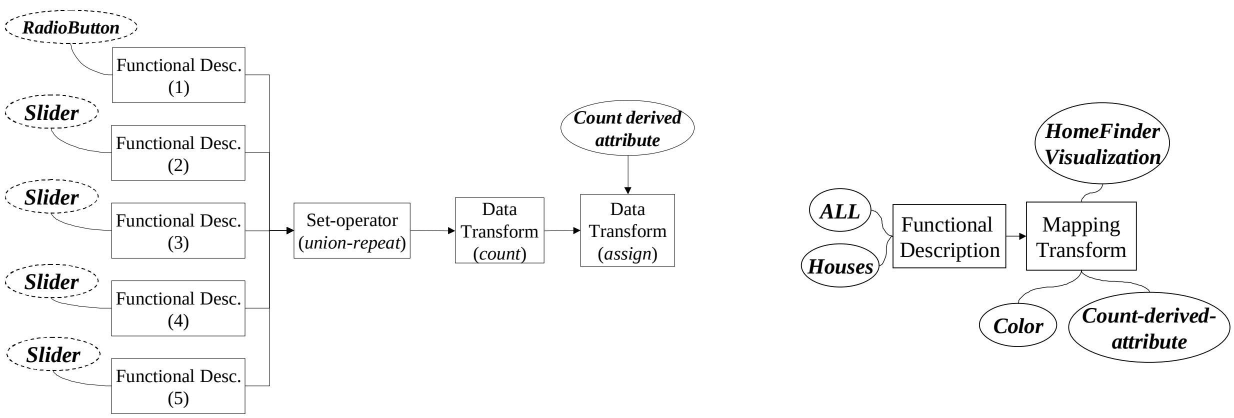


Figure II-11: HomeFinder system [*Tweedie, 1994*] (borrowed from http://infoeng.ee.ic.ac.uk/~lisat/LisaDir/att.html). There are five single-axis aligned charts and a *mark* in each chart represents a *house* data concept. This system uses object definition composition to integrate the object sets selected by the three sliders and the single set of radio buttons. We then *count* the number of times a house appears in the combined set, and use this *count_attribute* to set the color for a given *house* concept.

The type of object definition composition shown in Figure II-10 is quite common. It also occurs in the HomeFinder system [*Tweedie, 1994*] depicted in Figure II-11. In the HomeFinder system each house is represented by a *mark* in each of the charts. There are 5 charts, each with a different house attribute encoded on the *y-axis*. The *x-axis* shows the number of houses that have a particular attribute value. Users get to place constraints on different house attributes using *sliders* and *radio buttons*. The *marks* within each chart are then colored based on the number of constraints passed by the house data concepts they represent. The specification for this method is shown in Figure II-12a. There are several functional description operators in this diagram, each corresponding to a data attribute constraint that is controlled by the *sliders* or *radio buttons*. Each of these constraints defines a set of objects and these sets are combined together using the *union-repeat* set operator. The *union-repeat* operator is similar to the *union* operator except that duplicate objects are not deleted. The combined set is then passed through a *count* data transform that calculates the number of times a house concept appears in the input set. The results of the *count* function are then associated with each house data concept by using the *assign* data transform operator[2].

Using a separate mapping specification (Figure II-12b), we encode the *count* derived attribute with the *mark color* property in each of the charts within the HomeFinder visualization (Figure II-11). A separate specification is used here because the mapping parameters do not change based on changes in the threshold constraints (i.e. the *count* attribute is always mapped to *color* irrespective of changes in the threshold constraints). Combining the mapping transform with Figure II-12a would create a new mapping each time we change the house selection constraints, and this is not the effect we desire.



(a) In this specification we compute the *count-derived-attribute* which stores the number of times an element appears in the selection set produced by the *union-repeat* operator

(b) In this specification we map the *count-derived-attribute* data attribute, which we computed in Figure II-12a to the *color* graphical property in the *HomeFinder Visualization*.

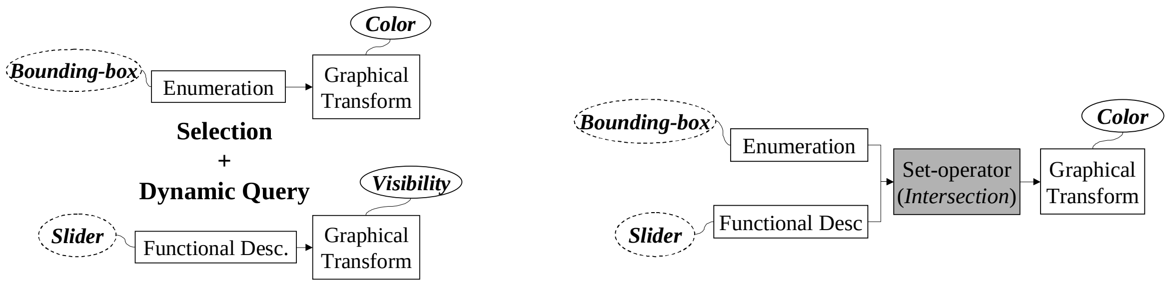
Figure II-12: Functional specification for the HomeFinder system shown in Figure II-11.

We can also apply object definition composition to primitive techniques that have different object definition methods. For example consider the dynamic query slider technique that uses functional description object definition and the object selection method that uses enumeration object description. The *ODT diagram* for both these

---

[2] We show the *assign* operator here so that we may illustrate how the *count* results change the *color* graphical mappings of the *marks* within the visualization. In general however, we leave out *assign* operators in most of the other technique specifications in this chapter because they do not play an important role in capturing the functional essence of a technique. In the next chapter where we discuss the visualization technique instantiation level, we detail all the instances where *assign* operators are added.

techniques are shown in Figure II-13a. We compose both these techniques using the *intersection* set operator as is shown in Figure II-13b. The technique shown in Figure II-13b only highlights objects that are both enumerated by the user through *bounding-box* selection and that fall within the functional constraints of the dynamic query slider.



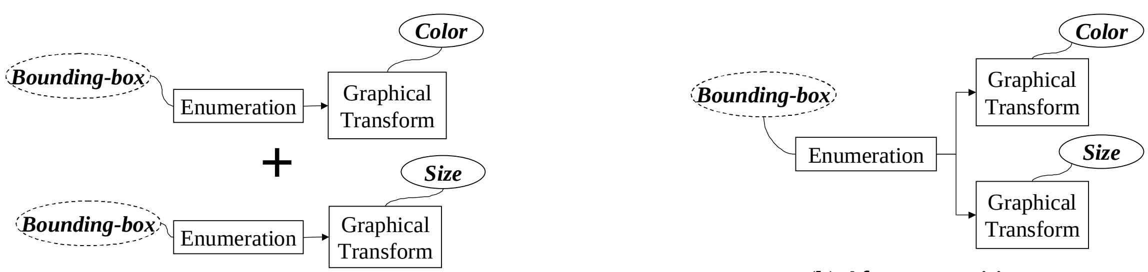
(a) Before composition.
The dynamic query technique allow users to search for data elements by placing constraints on a single data attribute by using a *slider* input-device. The selection method allows users to enumerate a set of graphical objects using a *bounding-box*. The selected objects will then be color-highlighted to show that it has been selected.

(b) After composition
(the object sets from both techniques are combined using the *intersection* set-operator)

Figure II-13: Object definition composition on the object selection and dynamic query slider technique. The resulting technique ONLY colors those objects that are selected within the *bounding-box* as well as passes the constraint set on the *slider*.

II-2.2 Transformation Composition

Unlike object definition composition, which combines two or more object definition sets and applies the same transformation(s) to the resulting set; transformation composition applies different transformation functions to the same object definition set. For example, consider the object selection technique outlined above (Figure II-13a). Suppose that in addition to highlighting the selected objects *red*, we also want to enlarge them. To achieve this effect we apply two different graphical transforms to the selected object(s) as is shown in Figure II-14b.



(a) Before composition

(b) After composition

Figure II-14: Transformation composition for two different selection techniques with different visual feedback effects. The resulting technique colors and enlarges the objects selected by the *bounding-box*.

We can also perform transformation composition on different classes of transform functions, e.g. a graphical transform and a data transform. For example in the simple aggregation technique described in Figure II-9, we indicate the objects that have been selected for aggregation by highlighting them, in addition to performing the data grouping. In this technique, transformation composition is applied to the graphical transform for highlighting the objects and the data transform for aggregating the objects as is shown in Figure II-15b.

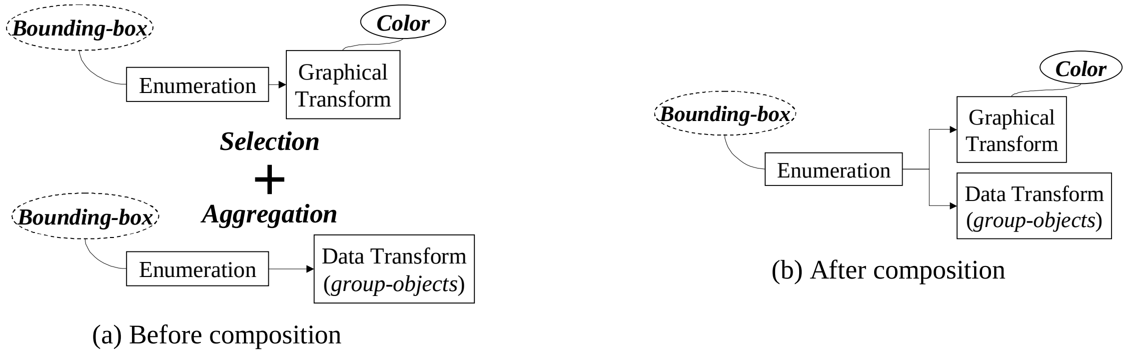


Figure II-15: Transformation composition for the simple aggregation technique shown in Figure II-9. Transformation composition is used here to combine a data transform for creating the aggregate object as well as the graphical transform that highlights the objects within the aggregate, *red*.

II-2.3 Produce-Consumer Composition

Producer-consumer (P-C) composition allows one technique (*producer*) to generate the arguments or information needed by another technique (*consumer*). An example is the *modified value-painting* technique [*Eick, 1992*]. This query method allows users to select a set of objects with an input-device. Chosen data attributes of the selected objects are then summarized and then used to functionally define and highlight another set of objects. The result of such a composition is shown in Figure II-16b where a set of objects is user enumerated through a *bounding-box*. A chosen data attribute (e.g. *selling_price*) of the object set is then summarized using the *mean* data transform. The calculated mean value is then passed to a subsequent functional description operator that selects all objects in the visualization with data attribute values less than the computed mean and then highlights them.

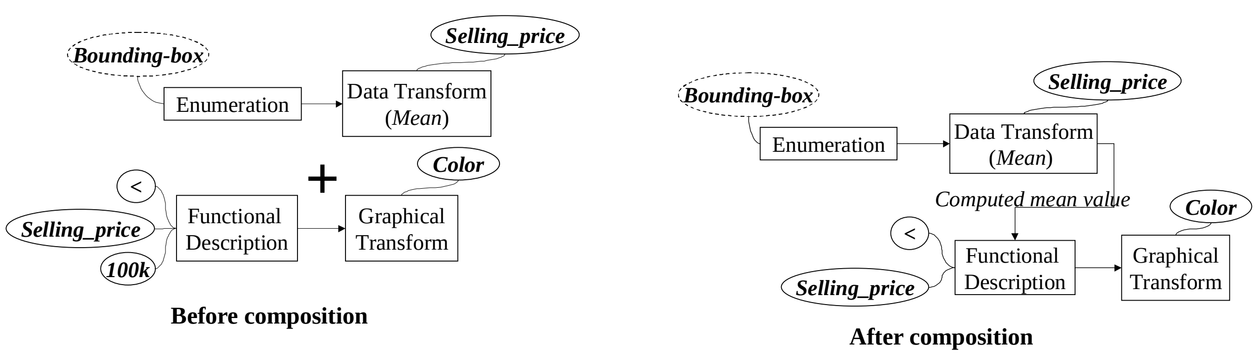
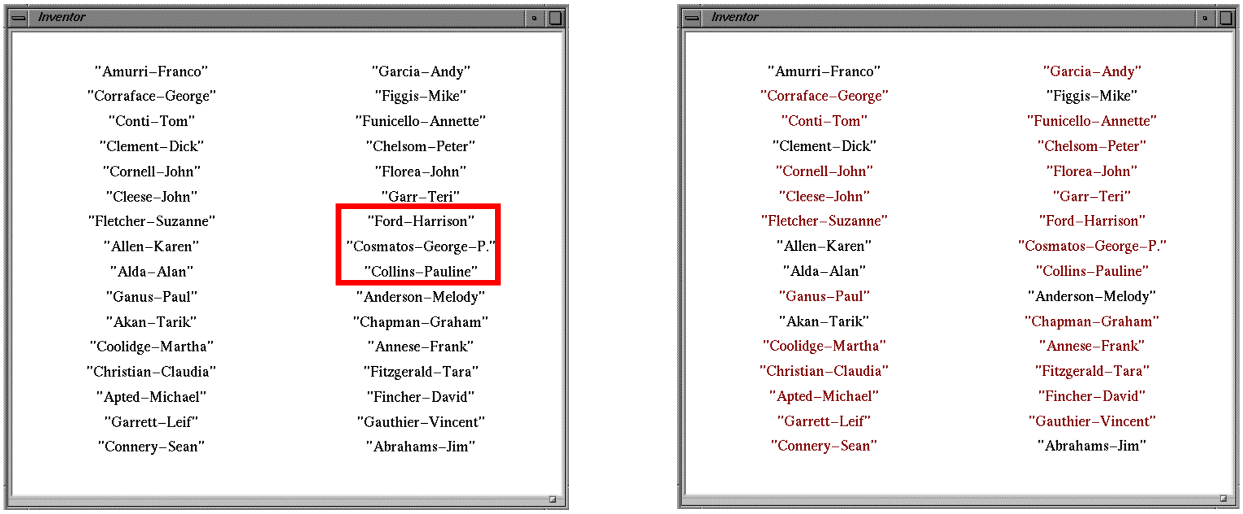


Figure II-16: P-C composition. The computed value from the producer technique (i.e. *mean selling_price*) is piped into the object definition component of the consumer technique and is used to select other objects in the visualization based on the computed *mean selling_price*.

For example, Figure II-17 shows the names of a set of *house_owners*. By using the *modified value-painting* technique, we may select a set of *house_owners* (*Ford-Harrison*, *Cosmatos-George-P*, and *Collins-Pauline*). Once selected, the technique computes the *mean selling_price* for the houses belonging to the selected *house_owvers*.

Finally the technique highlights all the selected *house_owners* as well as all other *house_owners* with houses costing **less than** the computed *mean selling_price*. Note that unlike the previous two composition classes, PC-composition techniques do not share common object definition sets nor transformation functions.



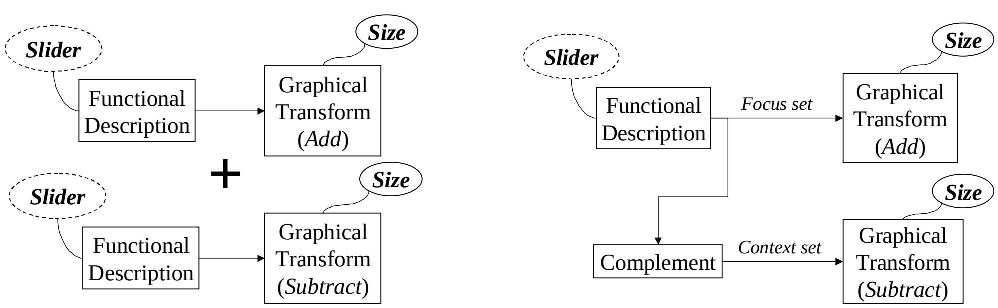
(a) ***Bounding-box*** **used to select set of** ***house_owners*****. In this example,** ***Ford-Harrison, Cosmatos-George-P,*** **and** ***Collins-Pauline*** **have been selected.**

(b) Selected ***house_owners*** **as well as** ***house_owners*** **with houses costing less-than the computed** ***mean selling_price*** **of the selected** ***house_owners*** **get highlighted** ***red*****.**

Figure II-17: P-C composition applied to the modified value-painting technique. ***Text*** **here encodes** ***house_owners*****.**

II-2.4 Partition Composition

The final class of composition, *partition composition*, is applied when the object definition component of a visualization technique generates more than one set of objects, and we want to transform each generated set differently.



(a) Before composition **(b) After composition**

Figure II-18: Partition composition applied to the dynamic query slider technique. Here we use partition composition so that we can enlarge the ***focus*** **objects (i.e. the objects that pass the** ***slider*** **constraint) and simultaneously contract objects in the** ***context-set*** **(i.e. objects that did not pass the** ***slider*** **constraint).**

For example we may alter the dynamic query technique slightly so that it enlarges objects that pass the slider constraint and contracts objects that do not. In this case the object definition component generates two object sets, the *focus set* and the *context set* and each of these partition sets are transformed differently, i.e., the *focus-set* has its size increased and the *context-set* has its size decreased (Figure II-18). Other common ways for partitioning sets are described in Goldstein et al.[*Goldstein, 1994*].

II-2.5 Summary

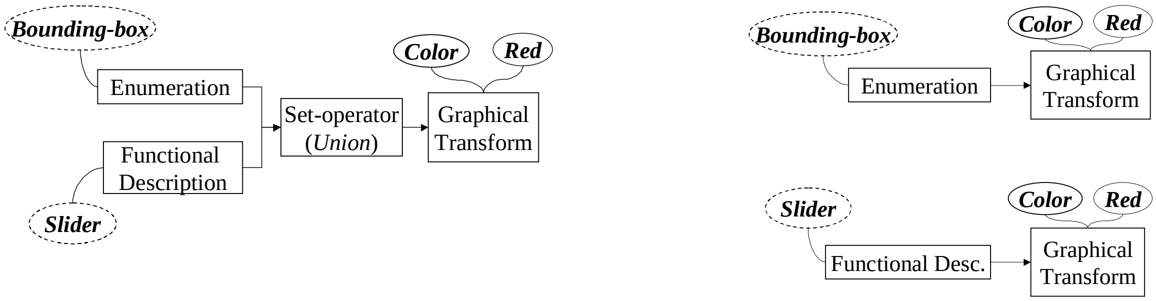
There are four ways in which primitive techniques may be composed. Each composition type is differentiated by the number and occurrence order of the object definition (OD) and transformation (T) components. Object definition composition, for example, can have multiple object definition components (*n* OD) followed by one transformation component (T); transformation composition has one object definition component (OD) followed by multiple transformation components (*n* T); P-C composition has multiple object definition/transformation components concatenated serially (*n* [OD + T]); and finally partition composition has one object definition component (OD), which generates *n sets* of objects which are subsequently transformed. This information is summarized in Table II-1 below.

Composition Type	
Object Definition	(*n*OD + T)
Transformation	(OD + *n*T)
Producer-Consumer	*n*(OD + T)
Partition	(OD + *n*[*set* + T])

Table II-1: Summarization of composition types

Even though the four different composition types are described separately in this section, we can apply multiple composition methods simultaneously within the same visualization technique. For example, we may use both object definition composition and transformation composition to apply different object sets to multiple disparate transformation functions. We may subsequently combine the technique through pc-composition and partition composition to additional object definition and transformation functions.

In some cases, the same visualization technique effects may be achieved both with and without composition. For example, suppose we want to highlight objects *red* either by selecting a set of objects through a *bounding-box* or through a dynamic query slider. One way to achieve this is to use object definition composition to union up the selected sets of both techniques and then highlight this combined set *red* (Figure II-19a). Another possibility is to leave both techniques separate as in Figure II-19b. The more appropriate specification depends on our intentions and our conceptualization of the technique. It is usually appropriate to compose two techniques if they are related in some manner, for example, if they are updated by the same input-device event. For example, suppose we want color highlighting to only occur on a *bounding-box* release event, then we would compose the two techniques because they are both triggered by the same input-device and we want to capture this relationship. On the other hand if the two techniques are not conceptually related, then they should be expressed separately. Composing techniques that are not conceptually related is a misrepresentation of their functionality.



(a) Composed technique. Techniques should be composed when they are conceptually related.

(b) Non-composed technique. Techniques that are not related should have separate specifications.

Figure II-19: Two different technique descriptions that achieve the same effect. Both techniques highlight objects selected by the *bounding-box* or the *slider, red*.

This issue of multiple specification solutions extends beyond the simple choice of whether to compose visualization primitives or not. More generally we can sometimes arrive at identical technique functionalities by using different combinations of primitive functions and composition operators. Which design solution is most appropriate would depend on secondary goals such as:

- How does each solution fit with our conceptual model of the technique.
- Which design is more general.
- Which design is more computationally efficient.

Theoretically, compositions allow us to generate an infinite number of designs because we can keep adding more and more operators onto the visualization technique specification. Practically however, the space we are exploring is not infinite. Techniques are only useful when we transform the visualization data and graphical objects in simple and fairly well understood ways. Users have a limited area of attention and a limited amount of cognitive resources. If too much of a visualization is changed, users may miss out on many of those changes; or if the changes are too complex, users may misunderstand or misinterpret the results. In the next section we start exploring the visualization techniques design space using the *ODT model* developed thus far. From this analysis we will see that many common visualization techniques contain very few composition operators and some have none at all. Note that the subsequent section may be skipped if readers are only interested in the use of the *ODT model* with respect to automatic design. Section II-3 explores the generality and coverage of our visualization framework rather than it relates to automatic design.

II-3 Visualization Techniques Design Space

In this section we show the generality and applicability of our framework by using it to evaluate current visualization designs as well as to make meaningful comparisons across techniques in different visualization systems. Specifically we use our framework to evaluate, compare and classify a set of common visualization techniques, found in Card et al.'s compilation of current visualization systems [*Card, 1999*] and *IEEE's Symposium on Information Visualization* (Table II-2, Table II-4, Table II-6, Table II-8). Some of these visualization systems (e.g. the *TableLens* system) utilize multiple visualization methods, in which case, we represent each of the methods separately in our analysis. Certain other common visualization methods (*dynamic query sliders* and *painting*) appear in multiple visualization systems, in which case, we only describe these base methods once. It is also important to note that while some of the systems analyzed have novel ways of representing and mapping data to graphics, we do not capture these new graphical representation methods here. A representation framework for visualization has been developed previously by Mackinlay and Roth [*Mackinlay, 1986a; Mackinlay, 1986b; Roth 1990*] and our framework defines functional operators that operate on the objects within these previous frameworks. To capture new types of representations we need to focus on expanding the representation framework and that is not the focus of this thesis. Thus only functional techniques and **not** systems with new graphical representations are shown in our analysis tables.

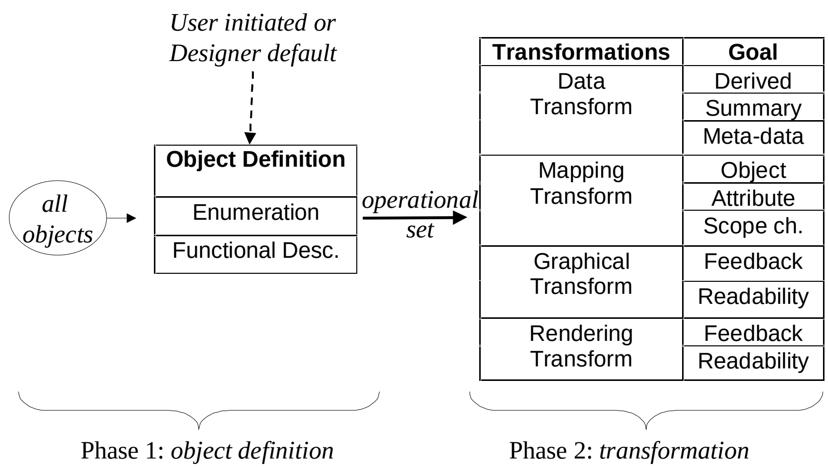


Figure II-20: The two components that form a primitive visualization technique – object definition and transformation

In the following sections we organize visualization techniques according to their main transform classes (i.e. *data transform* techniques are grouped together (Table II-2), as are *mapping* (Table II-4), *graphical* (Table II-6), and *rendering* (Table II-8) techniques). Some techniques may contain multiple transform classes in which case we repeat their specification in each group they belong to. Through this analysis we show that there are some common ways in which visualization techniques are currently used and combined. This knowledge will allow us to better understand visualization techniques and adapt them to more effectively fulfill our tasks and preferences. In addition, we can identify unexplored areas in the visualization techniques design space and start examining new forms of visualization methods. Finally, this analysis also illustrates that our framework is relatively general and is able to characterize a variety of techniques.

In these analysis tables each visualization method is represented by one or more rows. The specific object definition and transformation types of each technique is shown on the table columns. The first two columns represent the two object definition alternatives and the last four columns represent the four transformation classes. A visualization technique is represented by a set of connected highlighted (gray) cells. Highlighting occurs according to a technique's object selection and transformation functions. For example, the dynamic query slider technique has a functional description object definition component followed by a graphical transform that changes the visibility of the objects. As such it has a highlighted cell on the functional description column in Table II-6 that is connected to a highlighted cell in the graphical transform column. The foreground tool, on the other hand, lets users enumerate objects through mouse clicks, thus it has a highlighted cell in the enumeration column of Table II-6. This enumeration cell is connected to a highlighted cell in the graphical transform column, which represents the foreground function that brings selected objects up to the top of a graphical scene. Compositions are represented by double-lined arrows, with a symbol next to it indicating the composition type (OD = object definition composition; T = transformation composition, PC = producer/consumer composition; and PT = partition composition). The dotted arrows in the analysis tables represent *update-links*. *Update-links* indicate changes in a technique that is brought on by changes made in a different technique. For example the HomeFinder system in Table II-2 has an update link from its data transform function to a mapping function. This is because the data transform function updates the *count-derived-attribute* data values that are mapped to graphical objects in the visualization. Thus any change in those *count* values will result in a subsequent update to its graphical value mappings (refer to section II-2 for details). Note that this analysis simplifies the visualization techniques and only includes important functional features, so as to reduce diagrammatic complexity. As a result some of the techniques shown in Table II-2, Table II-4, Table II-6, and Table II-8 may not contain all of the visualization functions from their more complete specifications shown previously in this chapter.

II-3.1 Data Transforms

Data transform techniques are shown in Table II-2. It is perhaps not surprising to note that all these techniques either compose their data transform function with another transform class (*modified-value-painting, generalized-fisheye, simple-aggregation*) or have a subsequent *update effect* (shown as dotted lines) to another transform class (*HomeFinder, TableLens-sort*). Data transforms are commonly connected to a transform function of a different class because data transforms only generate non-visual results, and these results must somehow be shown to users. From Table II-2 we see that data transforms are commonly linked with mapping transforms (*simple-aggregation, TableLens-sort, HomeFinder, generalized-fisheye*). While the *modified-value-painting* technique has a pc-composition link to a graphical transform, there are no techniques with transformation composition links from a data transform to a graphical transform or a rendering transform.

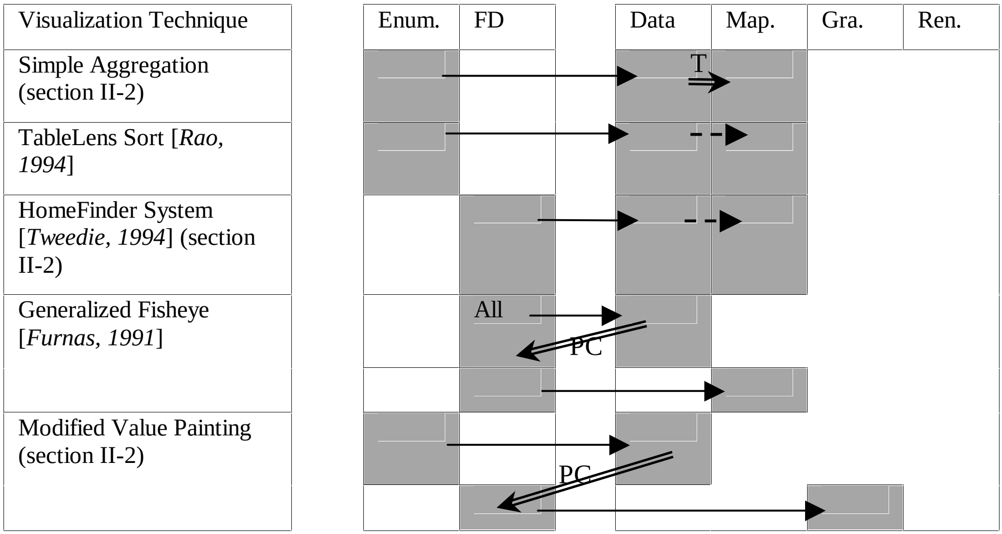


Table II-2: Data visualization techniques

An example technique that fills the data transform to graphical transform space, is a variation on the selection technique, which we call the *load-sensitive-selector*. This technique allows us to pick a set of data concepts, then highlights the graphical representations of those concepts according to the size (number of concepts) of the selected set. For example, a larger selected set will result in a more saturated highlight color while a smaller selected set will result in a lighter, less saturated highlight color. In this case we have a user-enumerated technique that is linked to a data transform for calculating the number of objects in the set. This count-value is then fed into a color graphical transform for computing the new highlight color. Yet another alternative is to use this count value to change the magnification factor of a rendering transform so that we automatically zoom in when we select a few objects and automatically zoom out when we select many objects. This produces a data transform to rendering transform technique.

Thus by looking at areas not covered by current techniques, we can derive new methods that may be useful for some task classes. The load selection technique for example can be used as a cue to indicate the existence of occluded objects in the selected set. If the saturation is high even though the number of objects that is selected appears to be small then there are probably some occluded objects that have been selected. Note that the new techniques that we discuss in this section (section II-3) are all manually designed based on our analysis of the visualization techniques design space using our framework. In addition these examples tend to be incremental in that they enhance an existing technique in fairly well understood ways. This is so that the expansions can be more easily conveyed and the uses of the expanded techniques are more readily apparent In appendix A-2.2 we present a more complex example of technique expansions that are also based on our framework.

Data transform techniques can be divided into three classes based on the information type generated: 1) derived attributes, 2) summary values, or 3) meta-data. The *generalized-fisheye* technique **computes a**

derived set of *degree-of-interest* (*DOI*) values based on the importance of a data concept to the current task (data transform type-1). Objects with low *DOI* values are subsequently culled from the display with a mapping transform. The *modified-value-painting* and *simple-aggregation* techniques, on the other hand, both **generate summary values** from existing data (data transform type-2). The *modified-value-painting* technique computes the *mean* for a set of values and uses that information through pc-composition for querying other data. The *simple-aggregation* technique summarizes a set of objects into a single aggregate object and then maps the aggregate object into the visualization. Finally the *TableLens-sort* and *HomeFinder* systems both use type-3 data transforms. In *TableLens-sort* the data transform function is used to calculate the **order of elements** (meta-data) based on an existing attribute. The *HomeFinder* system uses a *count* data transform function to generate meta-data for capturing the number of query conditions passed by a set of data concepts.

Note that there are interesting differences between the meta-data generated by the *TableLens-sort* and *HomeFinder* techniques. The former is generating meta-data based on the original data values while the latter is generating meta-data based on a set of query results. A data transform that is attached to a user controlled functional description function (like the *HomeFinder* system) usually falls into the latter category. Query data transform techniques are commonly transient, because their purpose is to give users one-time feedback on the current query results. As the query changes, new query information is computed/summarized and the previous results are discarded. In addition, because we are using data transforms to get an overview of a user initiated query, we only use *summary* and *meta-data* transform functions (e.g. to calculate the size of query set, the number of duplicate elements within the query set, the spatial dispersion of objects within the query set, etc). This separation between *query* and *non-query* data transforms is shown in Table II-3.

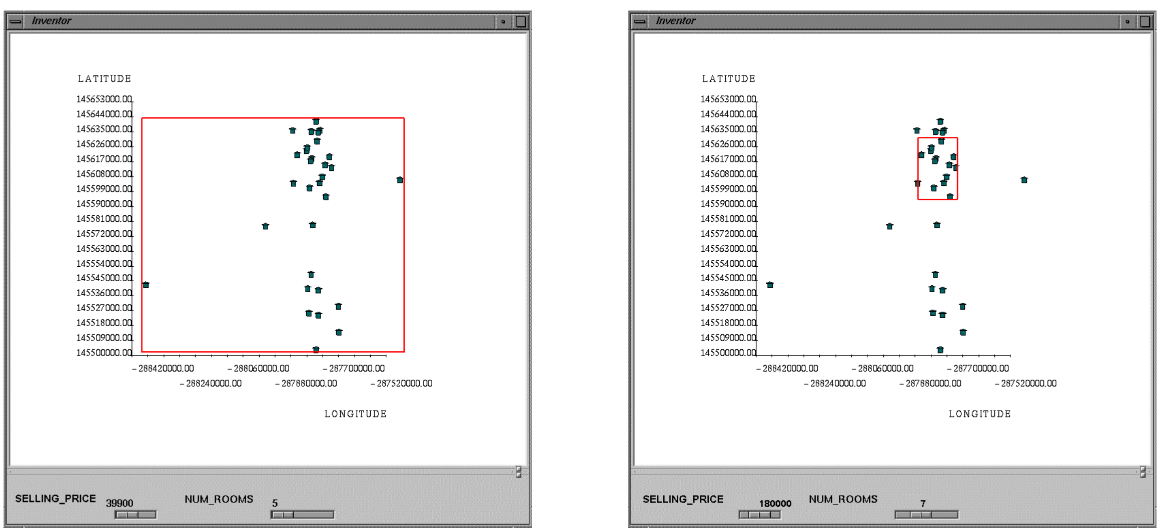
			Visualization Techniques
Query (transient)		Summary	
(data transform connected to a user defined functional description object definition method)		Meta-data	*HomeFinder*
Non-Query	Final results (persistent)	Derived	
		Summary	*Simple-aggregation*
		Meta-data	*TableLens-sort*
	Intermediate results (transient)	Derived	*Generalized-fisheye*
		Summary	*Modified-value-painting*
		Meta-data	

Table II-3: Goal categories of data visualization techniques

Non-query related data transforms are divided into two groups: a) those that compute final results, and b) those that compute intermediate results. Intermediate result techniques use data transforms to calculate temporary values that are used in subsequent functions. Thus, intermediate-result techniques are always linked either through pc-composition or transformation composition to other functions. An example is the *modified-value-painting* technique that uses a *summary* data transform to get the *mean* of a value set. This mean value is subsequently used in a functional description function. Another example is the *generalized-fisheye* technique that computes a set of intermediate *DOI* values, which are subsequently filtered with a functional description operator. Note that intermediate operations are also commonly used in complex computations. For example, to get the result for (A – B) + (C – D), we must first calculate (A – B) and (C – D), which are both intermediate results. These results are then fed into an addition data transform that produces the final result.

Final-result techniques, on the other hand, have more persistent output values that are commonly reused multiple times in several different tasks. In this case we are extending our database with new information. In the other two cases (i.e. query and non-query-intermediate) we are not looking to extend our database but rather just to summarize feedback results in a way that is easily assimilated by users or to fulfill intermediate tasks. Table II-3 show how the data transform techniques in Table II-2 may be classified based on these different classes of data transform goals.

In Table II-3 we see that none of the techniques considered fall into the query-summary category. One example technique we manually designed that fills this space is the *range-dynamic-query* technique that presents end-users with the spatial range of objects that fall within the query set.



(a) The *red bounding-box* encapsulates all the objects within the visualization because no constraints are currently set.

(b) The *red bounding-box* now only encapsulates objects that are on the top-mid portion of the display because only objects in that area pass the two *slider* constraints.

Figure II-21: Range dynamic query technique. In the example visualizations above, each *mark* represents a *house* data concept. The *x* and *y positions* of the *marks* corresponds to the geographic location of their respective *houses*. Objects are selected here by setting constraints using the two sliders at the bottom of the interface, which allow users to set threshold constraints on the *selling_price* data attribute and the *num_rooms* data attribute. The red *bounding-box* is then drawn so that it encapsulates all of the selected objects (i.e. all objects that pass the *slider* constraints).

For example Figure II-21 shows a *range-dynamic-query* interface that allows users to select objects based on *selling_price* and *num_rooms* using two *slider* input-devices. The visualization encodes geographic location of the houses on its *x-axis* and *y-axis*. In this interface we show summary information for the query objects by drawing an *annotation-box* (*red* rectangle) around all houses that fall within the query set. This is achieved by using the *get-max* and *get-min* summary data transforms to get the minimum and maximum *longitude* and *latitude* of the query objects. These data transform functions are later connected to a graphical transform function for changing the size of the *annotation-box*. From Figure II-21 we see that there is a relationship between house *selling_price* and *num_rooms* with *location*. The more expensive larger houses seem to be located in the top middle portion of the area. This technique is useful in cases where the visual display is large and it is difficult to focus in on the selected objects. By using the range dynamic query slider, we can automatically zoom in on the display based on the range of the selected set. This would work well in those cases where the query result is highly correlated with object positions in the visual display.

In Table II-3, the non-query-final-derived and non-query-intermediate-meta-data categories are also empty. However, all non-query data transform techniques (including the two empty ones in Table II-3), can commonly be performed in spreadsheet and data analysis systems which allow users to compute a wide range of non-query results based on user input formulaic expressions.

II-3.2 Mapping Transforms

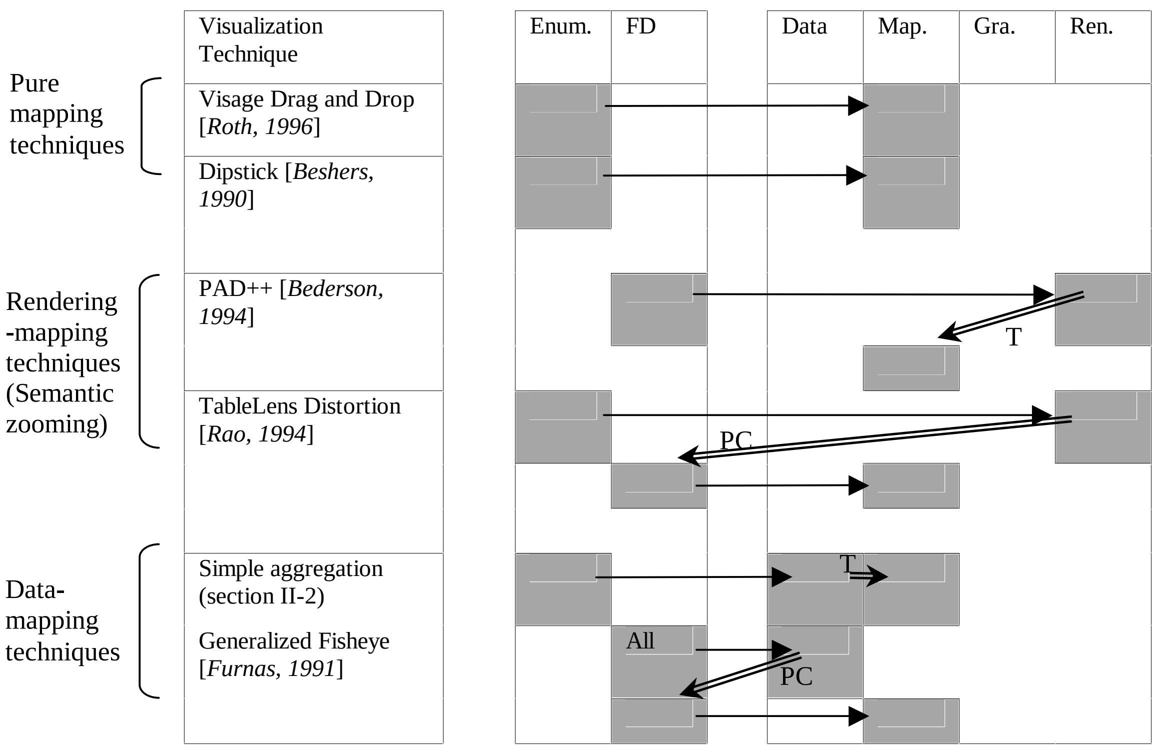


Table II-4: Mapping visualization techniques

The mapping techniques we analyzed are shown in Table II-4. Mapping transforms, as was discussed previously, are commonly associated with data transforms (*generalized-fisheye*, *aggregation*, *etc*). While there are several *semantic zooming* techniques (*PAD++* and *TableLens-distortion*) that compose mapping transforms with rendering transforms, there are no techniques in Table II-4 that combine mapping transforms and graphical transforms. An example technique we designed that fills this space is the *information-enhancer* technique that allows users to select a set of focus data concepts, and then increases the number of data attributes mapped (i.e. the amount of information encoded) for those chosen concepts (mapping transform) (e.g. similar to semantic zooming techniques). At the same time however, we also increase the size of those objects (graphical transform) so that the additional information encoded can be viewed more clearly and accurately. The *TableLens-distortion* and *simple-aggregation* techniques in Table II-4 can also be expanded with a graphical transform for *coloring* the operational set (i.e. the objects being transformed) so that they appear more salient to users.

Mapping transforms are used first and foremost to show end-users the information required for solving their task(s). All visualizations use mapping transforms for this purpose. Some visualization techniques also use mapping transforms to change the content of visualizations in order to improve readability[3]. The contents of a visualization can be set or changed by 1) mapping data concepts to graphical objects, 2) mapping data attributes to graphical properties, or 3) changing the scope of existing mapping functions. Table II-5 shows how the mapping transform techniques shown in Table II-4 can be classified based on these goals.

		Visualization Techniques
Show task data	Object	*All visualization systems*
	Attribute	*All visualization systems*
	Scope	*Visage-drag-&-drop*
Readability	Object (*semantic zooming*)	*TableLens-distortion, PAD++*
	Attribute (*semantic zooming*)	*TableLens-distortion*
	Scope	*Visage-drag-&-drop, Dipstick, Generalized-fisheye, Simple-aggregation*

Table II-5: Goal categories of mapping visualization techniques

[3] Readability refers to problems that arise due to constraints of the visualization output media and constraints of our perceptual system. These problems reduce the effectiveness of a given visualization design because of object occlusion, ink density, lack of information presence, and dwarfed encoding scales.

Visualization *readability* can be improved through *semantic zooming* techniques as well as *scope* techniques. Semantic zooming techniques improve the readability of a graphic by allowing users to view different pieces of information at different degrees of detail. This is achieved by using different object and attribute mappings at different instances. When data sets are large, it is not possible to show all the information in detail without overwhelming the user and overcrowding the available display space. One way to solve this problem is to represent the information with simple graphical objects that take up little space and only map the current focus information to richer graphical objects. Usually a change in graphical object representation is coupled with one or more attribute mapping changes as well. For example in the *TableLens-distortion* technique, we change the graphical representation used from *bar* to *text* (and *bar*) when the space available for showing the data is expanded. We also use the *label* graphical property in addition to *bar-lengths* for showing the data values.

Scope techniques, on the other hand, use mapping transforms to change the data concept or graphical object set being shown in the visualization but not the graphical representation class or graphical properties used, as was done in semantic zooming. The *visage drag and drop* technique [*Roth, 1996*] is a scope technique that allows users to select and transfer sets of objects from one visualization to another by changing the set of data concepts being encoded within each visualization. Operations available in the "*Data source*" menu of a *Microsoft Excel* chart uses mapping transforms in much the same way. To add objects into a chart users select a set of data concepts from the spreadsheet and then choose the "*Add*" option in the "*Data source*" menu. Both of these scope changing techniques can be used to improve visualization readability only showing the relevant data concepts at any given instance, thereby reducing clutter and ink density. However, using mapping transforms in this way to solve readability goals requires that we keep track of how we changed the contents of the visualization so that we can restore deleted information that may be needed at some later point of our task. In contrast, the *generalized-fisheye-lens* technique automatically changes the scope of visualizations based on a *degree-of-interest (DOI)* function and thus does not require users to manually control the object scope, but automatically adds and deletes objects as necessary. As a result, unlike the *visage-drag-and-drop* technique, we do not have to keep track of scope changes made to the visualization. However, it can be difficult to devise appropriate *DOI* functions for our task(s). Scope techniques are also often combined with data summarization operators (e.g. in certain aggregate methods) to add in new summary information and delete the original objects or values that have been summarized. This allows us to improve visualization readability because the number of objects shown are reduced.

II-3.3 Graphical Transforms

Graphical transforms (shown in Table II-6) are used to provide feedback or improve the readability of visualizations. In Table II-6, we see that graphical techniques are not commonly combined with other transformation classes. In the previous sections we had discussed several possible new design alternatives for combining data and graphical transforms as well as mapping and graphical transforms. Here we discuss designs that combine graphical and rendering transforms. Techniques do not usually combine graphical and rendering transforms because both of these function classes serve similar purposes, i.e. to provide feedback or to improve readability. In addition, rendering transforms may sometimes warp the effects of a graphical transform that operates on the same graphical property class (e.g. *positional*, *retinal*) making the resulting visual effect of the combined technique difficult to interpret. Thus a design that attempted to combine both graphical and rendering functions through transformation composition would do well to apply the graphical transform to different graphical property classes. For example, we could augment the *TableLens-distortion* technique (which operates on *positional* graphical properties) with a graphical transform that colors all objects within the lens *red* (i.e. operates on *retinal* graphical properties). This highlights the selected cells so that users can more easily tell which objects are being expanded. The resulting specification has mapping, graphical, and rendering functions combined together through transformation composition. Another alternative is to combine graphical and rendering functions using pc-composition. For example, to reduce occlusion, we could render all graphical objects that exceed a threshold *area* as wire-frame objects instead of solids. This technique uses a graphical transform to compute the *areas* for all graphical objects within the visualization. The *area* values are then piped into a functional description object definition function that ultimately leads to a wire-frame rendering transform.

	Visualization Technique
Feedback	Dynamic Query [*Ahlberg, 1992*]
	Value Painting [*Eick, 1992*]
	Modified Value Painting (section II-2)
	Simple selection (section II-2)
	Painting [*Becker, 1987*]
Readability	Foreground Tool
	SDM general [*Chuah, 1994*]
	SDM distance [*Chuah, 1994*]
	Shrimp [*Storey, 1997*]
	TableLens Column Move [*Rao, 1994*]
	Cone Trees [*Robertson, 1991*]

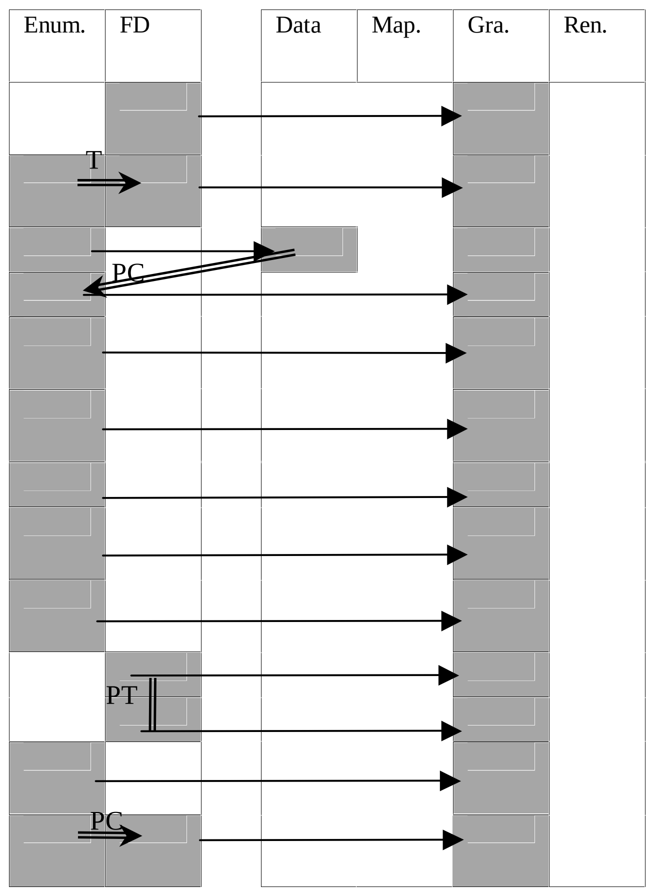


Table II-6: Graphical visualization techniques

There are two types of graphical technique feedback as is shown in Table II-7: search feedback and internal state feedback. Search-feedback techniques use graphical transforms to make the results of a query more salient so that a user's attention is drawn to those objects. Some example search techniques include *dynamic-query-sliders*, and *value-painting*. All of these techniques allow users to specify a functional description of the data concepts they are interested in. The appearance of the graphical objects representing the specified data concepts are then colored, made visible, enlarged, etc, to give feedback to users on the results of their search. The graphical transforms used for search and internal-state feedback are often simple such as setting a group of graphical values to a constant. This is however not a requirement. In appendix A-2.2 we consider a hybrid search technique with complex feedback effects involving many graphical transform operators.

		Visualization Techniques
Feedback (*commonly simple feedback)*	Search feedback (*functional description object definition*)	*Dynamic-query-sliders, Value-painting*
	Internal state updates (*stored internal information that is not readily apparent*)	*Object-selection, Simple-painting*
Readability	Occlusion	*Foregound-tool, SDM-distance, Shrimp*
	Ink density	*Shrimp*
	Information Presence	*TableLens-column-move, SDM-distance*
	Dwarfed Encoding Scales	*SDM-size-general*

Table II-7: Goal categories of graphical visualization techniques

Internal-state-feedback techniques use graphical transforms to encode internal state information and relationships among objects that may not be apparent in the visualization. *Object-selection* and *simple-painting* are some example internal-state-feedback techniques. For object selection, feedback is required to give users persistent state information on the object set that he/she is currently controlling. This is especially important when we have multiple object sets, or when we create an object set by slowly adding objects into it (many applications allow users to do this by using *shift-click*). In both these cases it is difficult to keep track of the objects that are currently selected, thus visual feedback is required to show users the current "*selection state*". The *simple-painting* technique, on the other hand, uses graphical transforms to reveal object relationship state-information. Specifically, graphical feedback is used to show the mapping relationships between certain data concepts and graphical objects (i.e. which graphical objects correspond to a given set of data concepts).

Readability graphical transform techniques change the effectiveness with which users can view the graphical elements that are already contained within a visualization. This is in contrast to mapping techniques that change visualization readability by altering visualization content. We focus on four primary readability problems: 1) occlusion, 2) ink density, 3) information presence, and 4) dwarfed encoding scales. These readability issues are described in greater detail in appendix F. Unlike feedback techniques, readability enhancing techniques such as *Shrimp*, *foreground-tool*, *SDM-distance-operation,* and *TableLens-column-move* tend to have an enumeration object definition component. This is because readability problems are often the result of complex spatial relationships among the objects and it is difficult to capture the set of objects involved, functionally. There are several different ways in which readability may be improved. The *foreground-tool* improves visibility by reducing occlusion for a set of interesting objects; the *Shrimp* system enhances the visibility of focus objects by making them larger in size, and reducing the density of elements around the focus objects (this may sometimes remove occlusion problems as well). The *SDM-distance-operation* and *TableLens-column-move* techniques address the information presence readability issue. The *SDM-distance-operation* allows users to bring a set of objects closer together and facilitate comparisons by transforming the objects spatially so that they fall on a straight vertical line perpendicular to the user's point of view. The *TableLens-column-move* technique, on the other hand, allows users to move table columns closer together so that the values are more easily comparable. Note that the *SDM-distance* technique also has the side effect of removing occlusion from the focus set, as well as enlarging the focus objects by bringing them closer to the user.

II-3.4 Rendering Transforms

Finally, like graphical techniques, rendering techniques (shown in Table II-8) are also used to provide feedback and improve visualization readability.

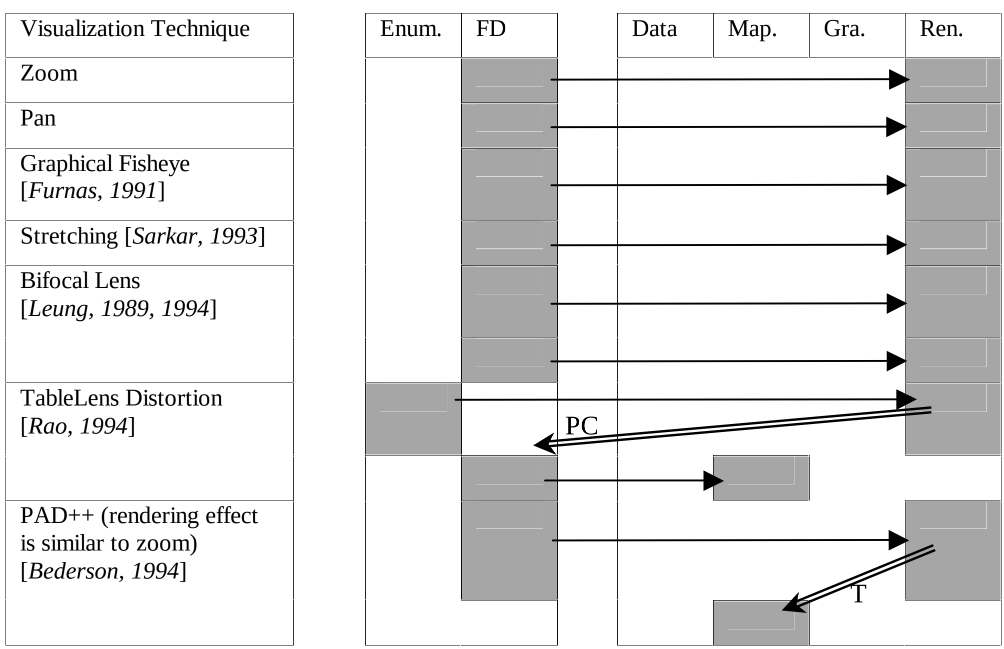


Table II-8: Rendering visualization techniques

Rendering techniques transform the way with which a visualization graphical scene is drawn on the CRT display, so that users may focus their attention on different parts of the scene that are pertinent to their current task. Because of the nature of such transforms, most rendering functions operate on the entire visualization window

[*Furnas, 1991; Leung, 1989,1994*]; transforming only sub-regions will cause discontinuities in the display space that are often distracting, and result in a loss of information [*Hollands, 1989*]. Example techniques that do not operate on the entire visualization window includes the *lens distortion* techniques [*Bier, 1994; Rao, 1994*].

As can be seen in Table II-9, rendering techniques are more commonly used to solve readability issues rather than feedback. One reason is that feedback techniques usually only require simple visual stimuli whereas rendering transforms tend to bring about more complex visual changes. There are two main classes of rendering techniques: *navigation techniques* and *distortion* or "*focus + context*" techniques. Navigation techniques such as *zoom, pan* and *scrolling* allow users to view different areas of the information space when the output media is not sufficiently large to contain all the information simultaneously (i.e. information presence readability issue). Note that as is shown in Table II-9, *zoom* techniques can be used to solve other readability problems as well. The main problem with *navigation* techniques, however, is that users may easily lose track of where they are in the information space.

			Visualization Techniques
Feedback			None analyzed
Readability	Navigation	Occlusion (only *perspective occlusion*)	*zoom, pan*
		Ink density	*zoom*
		Information Presence (view different information)	*zoom, pan, scrolling*
		Dwarfed Encoding Scales	*zoom*
	Focus+context	Occlusion (only *perspective occlusion*)	*graphical-fisheye-lens, bifocal-lens, stretching*
		Ink density	*graphical-fisheye-lens, bifocal-lens, stretching*
		Information Presence (more information to be shown)	*graphical-fisheye-lens, TableLens-distortion, bifocal-lens, stretching*
		Dwarfed Encoding Scales	*bifocal-lens, TableLens-distortion*

Table II-9: Goal categories of rendering visualization techniques

Focus + context techniques or spatial distortion techniques, such as the *graphical-fisheye-lens, bifocal-lens, stretching* and *TableLens-distortion* techniques warp the visualization space so that less output space is given to the information on the periphery and more output space is given to the focus objects. By reducing the contextual areas, these techniques can be used to show more information at any one time compared to non-distorted displays (i.e. greater information presence) and as a result it is less likely that users will get lost in the information space. In addition, the expanded focus area(s) help reduce ink density around focus objects so that they can be viewed more clearly. In 3D-displays, these distortion techniques may also be used to remove *perspective-occlusion*. Note that even though *TableLens-distortion* is a focus+context technique, it does not appear in the occlusion and ink-density categories of Table II-9 because these two readability problems are not applicable to the *table* representation used in the *TableLens* system. Finally, focus+context techniques may also be used to expand dwarfed encoding scales (i.e. a positional axis that is too small in size to accurately represent all of the data values). However, their non-linear magnification functions distort the encoding scales, making them difficult to interpret. The two exceptions are the *TableLens* and *Bifocal-lens* techniques, which use very simple distortion functions. In these techniques the display only has two different distortion scales, one for the *focus area* and one for the *context area*; unlike the other techniques which apply a continuous distortion function. As a result any distorted axes will also only have a focus and a context section, and as such are easier to interpret.

II-3.5 Summary

In this section we have classified a set of common visualization techniques using the object-definition/transformation structure described in previous sections. Based on this classification we were able to draw some interesting similarities among the different transformation techniques and categorize them based on their goals as is shown in Table II-3, Table II-5, Table II-7, and Table II-9. We were also able to recognize the common ways in which current visualization techniques are used, how they are commonly composed with other functions, and the areas in the design space that have yet to be rigorously explored. In fact, in this section we presented some simple adaptations of current techniques (e.g. *load-sensitive-selector, range-dynamic-query, information-enhancer*) that fall within some of the less populated design areas. These techniques are interesting, if not in their end-use, then in filling out the visualization design space and in illustrating the strengths and weakness of new technique classes that have never been explored. Note that in this section we do not discuss how "good" a technique is at fulfilling its intended task. The goodness of a technique with respect to a task is evaluated in chapter IV-2, using four different measures: articulatory, functional, expressive, and observational distances. These four distances determine which visualization techniques are most appropriate for a task by estimating the amount of motoric, cognitive, and perceptual effort users must expend to solve their tasks.

II-4 Conclusion

In this section we decomposed visualization techniques into two components: an object definition component and a transformation component. In the object definition component users pick a subset of data and graphical elements from all the available elements in the visualization system. In the transformation component, different transform functions are applied to the objects from the object definition component in order to bring about different visualization effects. These simple two-component primitive visualization techniques may be composed with each other using four different classes of compositions: object definition composition, transformation composition, producer-consumer composition, and partition composition. Each of these composition types are characterized by different patterns of object definition and transformation chains. Based on the *ODT model* (object definition/transformation) and composition functions, we get a better idea of what constitutes a visualization technique, how they are built, combined and used. This is crucial to our automatic design system and we show how the concepts set forth in this chapter are applied to our prototype design system in chapter V. In addition, we can also use our framework to scope out the design space for common interactive techniques (Table II-2, Table II-4, Table II-6, and Table II-8) and make comparisons across techniques from different visualization systems. These tables show the flexibility of our framework in being able to represent a wide range of current visualization techniques. It also reveals the ways in which common techniques are used and combined so that we are better aware of current design boundaries and where the unexplored areas are within the visualization techniques design space.

This chapter focussed on examining visualization techniques at a higher, functional level of abstraction where we are mostly concerned with the general class of object definition and transformation functions used, as well as their goals. There is little discussion here of inputs and outputs to the functions, how these functions get their needed input arguments, and how input-devices can be integrated into the design. These issues must be solved before we can use our visualization techniques framework in our automatic design system to render or generate a working visualization interface. We consider these instantiation issues in the next chapter.