Appendix A

Appendix to Functional Visualization Techniques Framework (Chapter II)

A-1 ODT Diagrammatic Notation

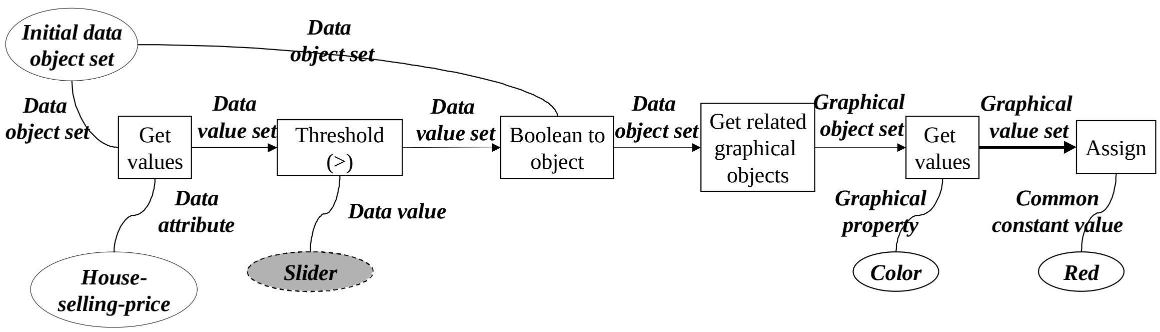


Figure A-1: Example ODT diagram for the dynamic query slider [*Ahlberg, 1992*] visualization technique

All function primitives are shown with normal *Times-Roman* font within rectangles and all inputs to the primitive functions are shown as ***italicized bold*** text within ovals. Inputs provided by users are shown with dotted ovals and those provided as designer defaults are shown with regular unbroken ovals. The directed arrows (→) connecting one primitive function to another indicate a flow of objects or values from a source function to a destination function. Arrows are sometimes also used to indicate temporal sequencing (i.e. a given primitive has to be executed before another). Temporal sequencing connections are different from regular connections because there is no information flow from the source to the destination function. Compositions can be achieved with regular connections or temporal connections depending on whether the operations have dependencies that require them to be ordered serially or whether they can be performed in parallel. In this thesis we do not differentiate between regular and temporal links because their differences do not have any significance or impact on our work.

A-2 Exploring the Space of Visualization Techniques

In chapter II-3, we outlined some simple visualization technique adaptations and expansions that can be made to current techniques to fill in less populated areas in the visualization design space. In this section we show two complete examples of how existing techniques may be combined and varied. Not all combinations will be interesting or useful and we can identify the bad cases through common sense, general design goodness measures (chapter IV-2), or user testing.

For each of the two examples we will bring together a pair of techniques that serve different functions, and the resulting composed technique will encapsulate functionalities from both of the base methods. The first example (section A-2.1) combines: 1) the *highlight object selection* method, which allows users to select a set of objects and then colors the objects to show that they have been selected, and 2) the *dynamic query slider* method, which allows users to search for a set of objects based on specific data attributes. This example is simple and meant to illustrate how we can go about combining different functional components of existing techniques to form new behaviors.

The second example (section A-2.2) combines: 1) the *SDM distance operator* method, which improves the legibility of objects by bringing them closer together to ease comparison, and 2) the *HomeFinder* system, which allows users to search for a set of data concepts based on several constraints on their attributes. The second example explores an uncommon area in the visualization techniques' design space. As was described in chapter II-3, it is commonly the case that search techniques have simple feedback mechanisms, usually consisting of changing one graphical property in a straightforward manner (e.g. setting all the property values to a constant). In this second example we explore search techniques (e.g. *HomeFinder* system*)* that have more complex feedback methods, like the *SDM distance operator*.

A-2.1 Highlight Object Selection & Dynamic Query

Here we integrate the object selection and the dynamic query techniques using the composition rules described in chapter II-2. The highlight object selection technique allows users to select a set of objects through a *bounding-box* and then colors those selected objects *red*. The dynamic query technique allows users to define a set of objects by setting constraints on their data attributes. Constraints are set by using threshold functions (e.g. *greater-than, less-than, equal-to*) and the threshold value is determined by the user through a slider input device. Specifications for both these techniques are shown in Figure A-2.

Highlight object selection specification **Dynamic query slider specification**

Figure A-2: Highlight object selection and Dynamic query sliders

One way of combining the two techniques is to use pc-composition to pipe values acquired from the graphical transform of the highlight selection technique into the dynamic query technique as is shown in Figure A-3. Instead of using an *assign* graphical transform (*assign* sets the values of a data attribute or graphical property structure but does not produce any new results) as was done in the original highlight selection technique we can use a compute graphical transform. For example, we could apply the *multiply* graphical transform to the *width* and *length* of graphical marks (to calculate their area) and then feed these values to a functional description operator as is shown in Figure A-3. This allows us to select one graphical object and subsequently make all objects that are larger than the selected object invisible. This can be an interesting method to interactively reduce occlusion in a display. Note that all functions and inputs that have been changed or removed are indicated in Figure A-3 with a red cross and new functions and inputs are highlighted in gray. The problem with culling out graphical elements based on graphical object size rather than on task related data, however, is that we may accidentally remove data elements that are crucial to our task.

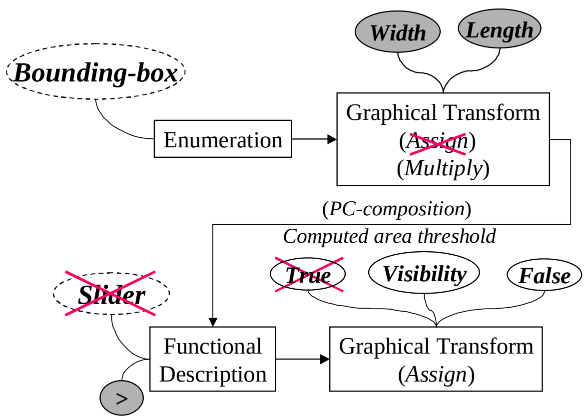


Figure A-3: P-C composition of selection and Dynamic Query

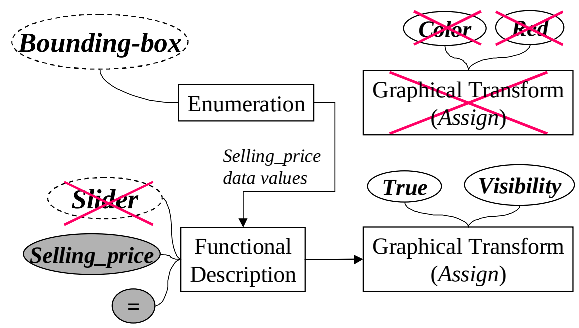


Figure A-4: Value painting specification

An alternative design applies the transform method to data values instead of graphical values. This produces a method like the value painting technique described by Eick et al.[*Eick*, *1992*]. Value painting allows users to select objects in one visualization with a *bounding-box*. We then search and highlight all other objects that have the same attribute values as the chosen objects (Figure A-4). In the value painting example, we bypass the graphical transform component and simply pass on the data values as is. The *modified value painting* technique (described in chapter II-2.3) is another design alternative that can be derived from applying pc-composition to the dynamic query and selection visualization techniques.

Another way of composing the two techniques is to use object definition composition (Figure A-5). In Figure A-5 we combine both object sets from the selection technique and the dynamic query technique together with the *intersection* set-operator so that only objects that are both selected with the *bounding-box* as well as fulfill the constraint set by the threshold *slider* are highlighted *red*.

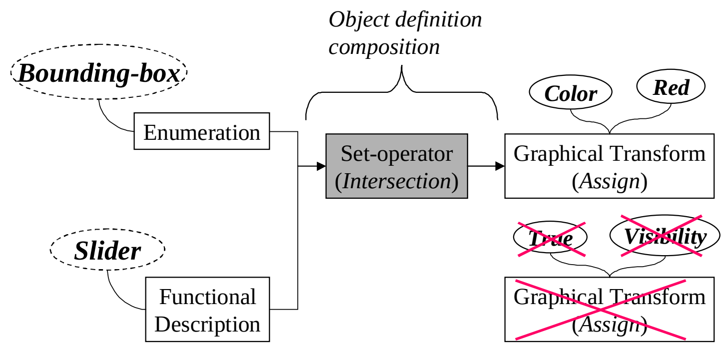


Figure A-5: Combining the object selection and dynamic query techniques using object definition composition

Another variation is to combine the graphical transform effects of both techniques with a transformation composition operator (Figure A-6). In this example, however, the two effects (*color-assign* and *visibility-assign*) do not integrate well together. This is because making non-focus objects invisible (as is done by the dynamic query slider technique) nullifies the use of the color highlighting graphical transform used in object selection. Since only focus objects are made visible, the color highlight effect is lost because all focus objects get highlighted in the same way.

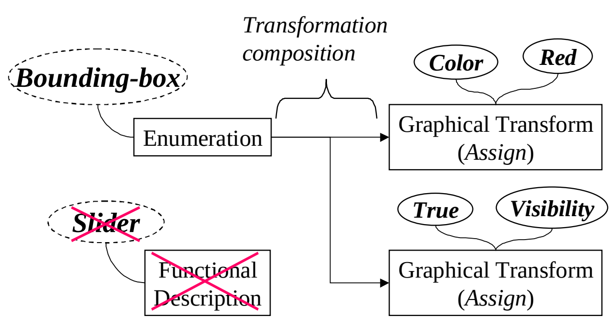


Figure A-6: Combining the object selection and dynamic query techniques using transformation composition

In this section, we see that combining the two techniques (highlight object selection and dynamic queries) allows us to integrate their different object definition and transformation methods.

A-2.2 HomeFinder System & SDM Distance Operator

In this example we explore a search technique that has more complex feedback mechanisms. Chapter II-3 showed that search techniques commonly use very simple feedback methods to show their results. For example, the HomeFinder system, the dynamic query slider technique, and the value painting technique all use simple color or visibility highlights to show the results of a search. An interesting exploration path is to see whether we could integrate a search technique with a richer graphical feedback technique that produced more interesting visual changes to objects within a visualization.

To pursue this path of exploration, we combine the HomeFinder search system with the SDM distance operator, which has rich visual feedback. The HomeFinder system was described in chapter II-2.1. This system allows users to set a number of functional description constraints on a set of data concept attributes. The graphical objects representing the data concepts are then colored based on the number of constraints passed by each concept (Figure A-7a). This is achieved through a *count* data transform that calculates for each object, the number of times it appears in a given input set. These count values are then assigned to the data concepts under consideration as a new attribute, i.e. the *count-derived-attribute*. This new attribute is mapped in a separate specification to *color* as is shown on Figure A-7b.

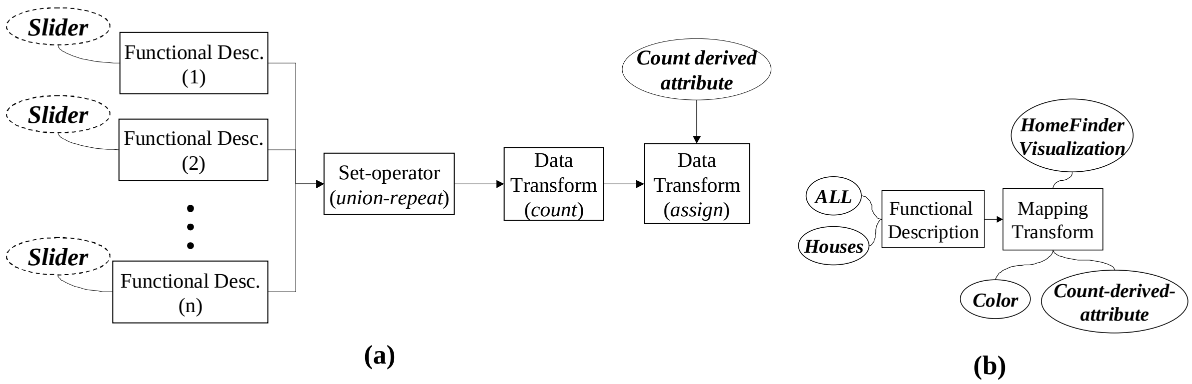


Figure A-7: HomeFinder system specification

The SDM distance operator improves the readability of a visualization by allowing users to move a set of objects to a user defined line of reference (shown in *red* in Figure A-8). By setting the line of reference to be close and orthogonal to our point of view, we improve our ability to compare object *size* or *height*, and also increase their *visibility*. This SDM distance technique is achieved by calculating for each object, the point on the reference line that is orthogonal to it (we refer to this point as the *reference point*) as is shown in Figure A-8. We then derive the distance from the original object positions to their *reference points* (i.e. *distance-to* attribute).

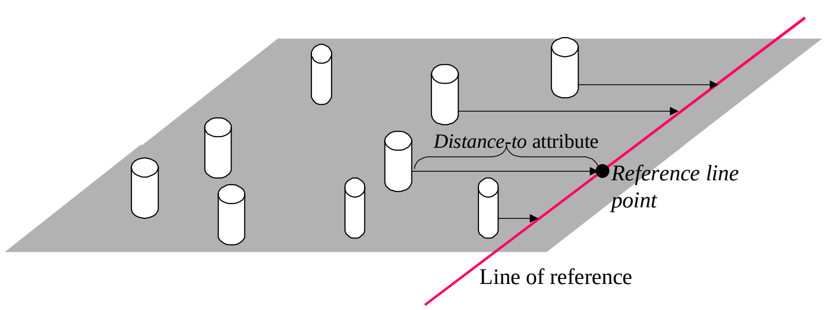


Figure A-8: SDM distance operator components: *distance-to* attribute, point of reference, and line of reference

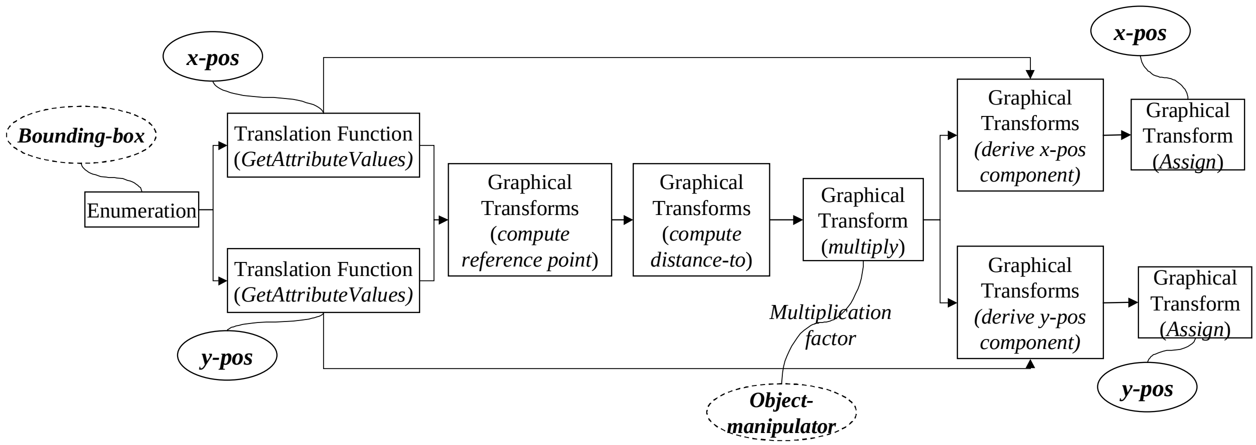


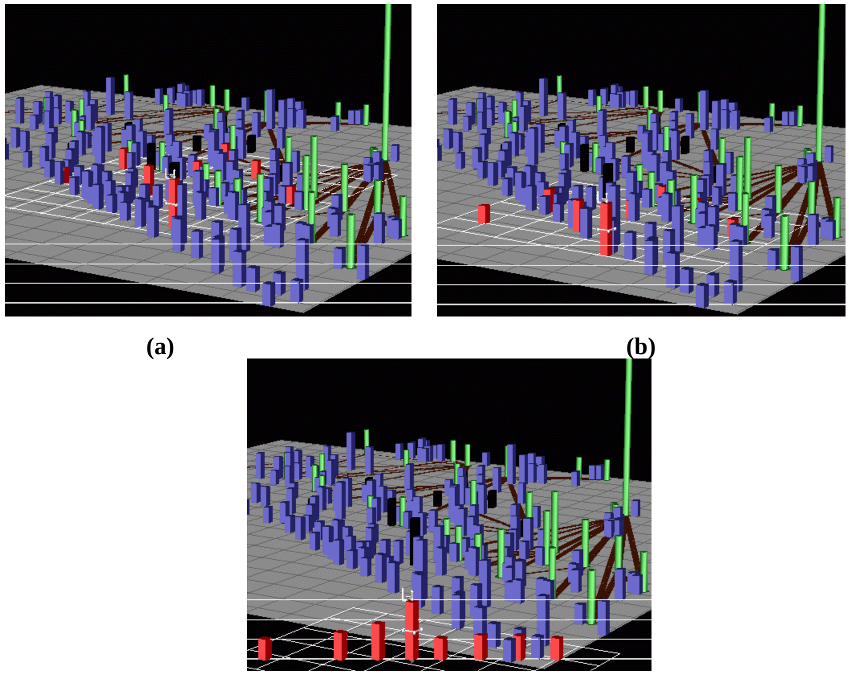
Figure A-9: SDM distance operator specification

Figure A-9 shows the design specification for the SDM distance operator[1]. We start by defining the set of objects we want to transform using the enumeration object definition operator. Multiple compute graphical transforms are used to calculate the *reference points* for these objects based on their original *xy-positions* and the position of the *line of reference*. Subsequently we compute the distances between the original object positions and their *reference points* (*distance-to* derived attribute). Users may then reposition objects anywhere along the orthogonal line between its original position and its *reference point*. Object repositioning is achieved by scaling the *distance-to* derived attribute through an *object-manipulation* input device that is tied to the *multiply* graphical transform that performs the *distance-to* scaling. Different multiplication factors cause new *distance-to* values to be computed. These *distance-to* values are then converted back into their *x* and *y* components and finally reassigned to update the objects' *x* and *y-positions*. Note that for simplicity, we represent certain sets of computation graphical transforms (*compute-reference-point, compute-distance-to, derive-x-pos-component* and *derive-y-pos-component*) with a single rectangle in Figure A-9 even though the actual operation consists of multiple simple graphical transform operators (e.g. *multiply, add, divide*).

By tying the multiplication factor to an input device, the technique allows users to slide a set of objects to and from the reference line. This enables users to maintain context of the objects' original positions. The sequence of images in Figure A-10 shows different *distance-to* multiplication factors and the virtual input device used to control those factors in the SDM system. When the *distance-to* scale is reduced to zero, all the objects get positioned along the line of reference as in Figure A-10c.

---

[1] Note that there are several *GetAttributeValue* translation functions in Figure A-9 which we have not yet described. We show these functions here however, to illustrate that the values being transformed are the *x-position* and *y-position* values of the objects. Translation functions are described in detail in chapter III-1.3, which also shows how and when such functions are added into the visualization technique specification.



(a) (b)

(c)

Figure A-10: Sequence of images showing different multiplication factors being applied to the distance-to attribute

By combining the HomeFinder technique and the SDM distance technique, we get an interesting synergy between their disparate goals. One way of combining these two techniques is to link the appropriateness of an object with respect to our search criteria to the distance of that object with respect to a reference element. I.e. we can interpret how good of a search match an object is by looking at its distance to a reference object. To do this we use the *count-derived-attribute* determined in the HomeFinder technique as multiplication factors for the *distance-to* computation in the *SDM* technique as in Figure A-11. Other alterations include removing the *object manipulation* input device from the SDM distance technique and applying the *distance-to* calculations to *all* the graphical objects in the HomeFinder visualization rather than just to a user enumerated set as was the case previously. In this way, the *count-derived-attribute* determines the percentage distance of every object to the *reference line*.

We can apply this technique to objects whose positions are already encoding values or to representations where the object positions are fully determined by the search results. In the former case it is important to note that the initial distance between each object and the *reference line* acts as an importance weight. Objects that are close to the reference line assign less importance to the *count* multiplication factors while objects that are farther away assign greater importance to them. For an object close to the reference line, even a high percentage change from its original distance would translate to a relatively small absolute position change. As a result objects that are far away from the *reference line* need to pass more selection constraints than objects that are closer to get to the same *distance-to* value.

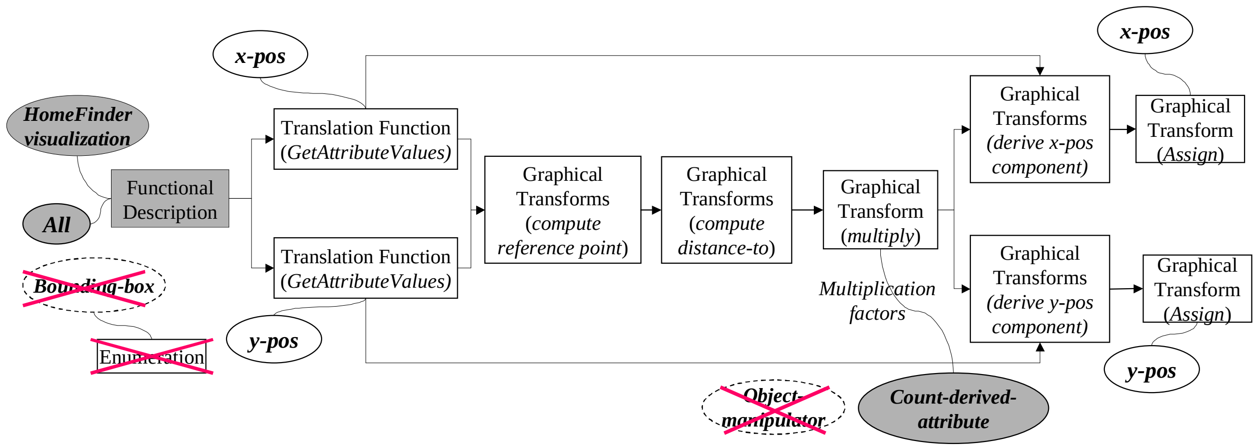
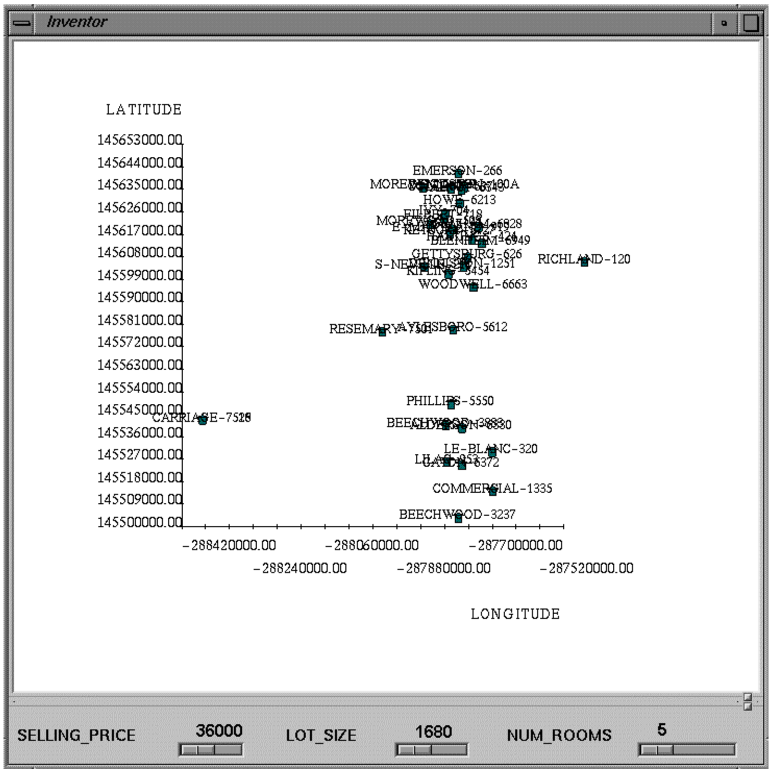


Figure A-11: Changes made to SDM and HomeFinder specifications

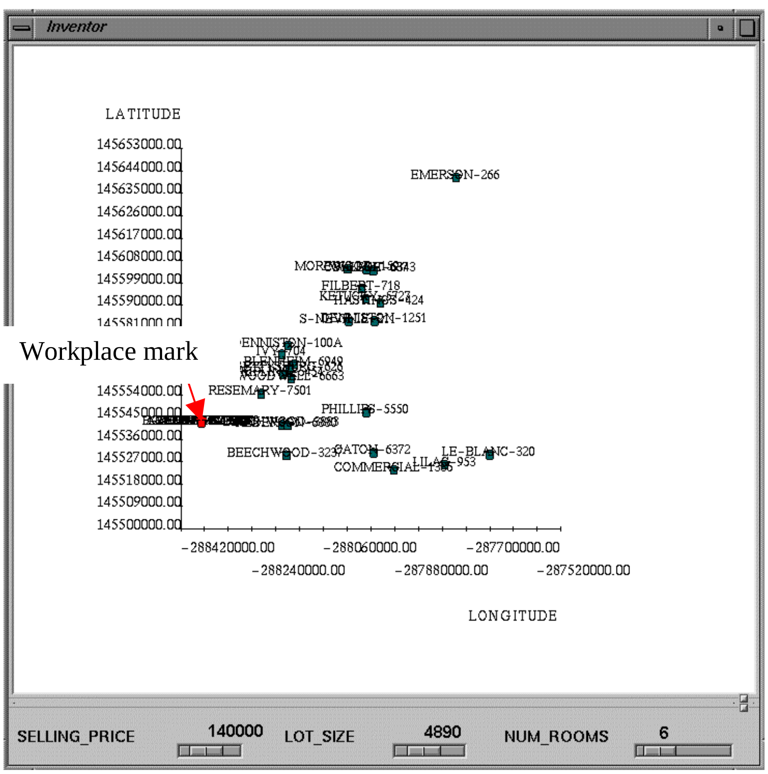
A task that is particularly appropriate for this case is one where there are natural weighting data attributes that can be easily mapped to spatial positions. For example consider the task of buying a house and suppose that we want the house to be as close to our workplace as possible. Houses that are farther away from our workplace will only be attractive if they fulfill many of our other house selection constraints such as the *num_rooms* in the house, the *selling_price*, the *crime-rate* in the surrounding area, the *availability of schools* and *hospitals*, etc. In this case, instead of using a reference line, we use a reference point, situated at our workplace (Figure A-12).

The *x* and *y* positions of the marks within the map in Figure A-12 are used to encode the *longitude* and *latitude* position of the houses that we might be interested in buying. This interface also has a set of sliders that allow us to set different house selection constraints (i.e. *greater-than selling_price*, *greater-than lot_size*, *greater-than num_rooms*) which will in turn change the percentage distance-to value of house concepts to our workplace mark. We are ultimately interested in those houses that appear closest spatially to the *red* mark. These are either the houses that are geographic neighbors of our workplace, or the houses that are farther away geographically but fulfill many of our other house selection constraints. In Figure A-12a, no workplace has been chosen, thus the position of the houses are their *longitude* and *latitude* positions.

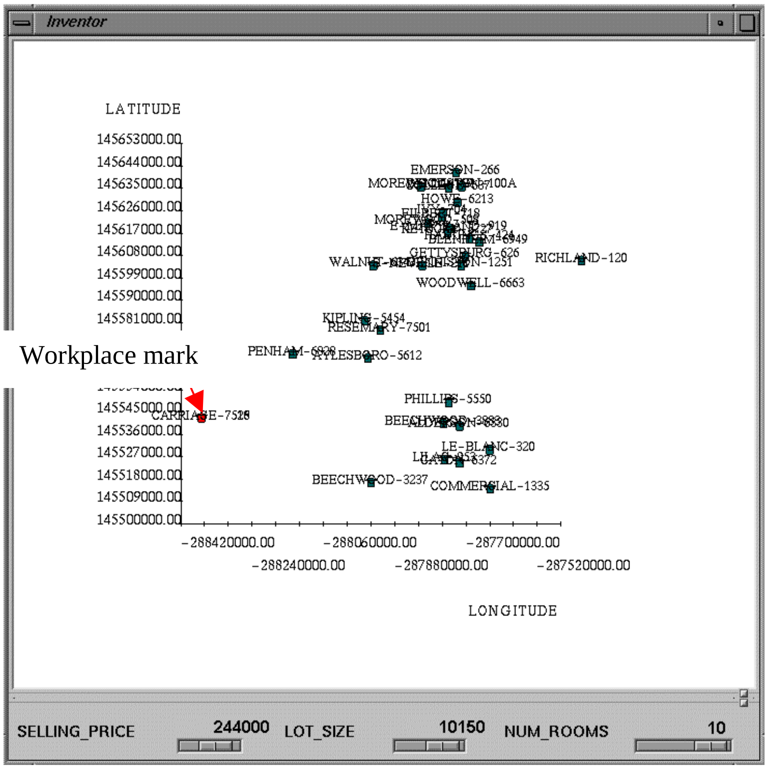
In Figure A-12b, a workplace has been selected and many of the house concepts gravitate significantly towards the *red* mark because the constraint conditions are less stringent (i.e. lower thresholds) and as a result most data concepts pass a significant number of the constraint conditions. In Figure A-12c, the threshold constraints are set higher and as the result the houses gravitate less towards the workplace mark.



No workplace chosen. Positions represent actual *latitude* and *longitude* values



(b) Low thresholds, greater proximity to workplace mark



(c) High thresholds, less proximity to workplace mark
Figure A-12: Example house selection technique

A problem with this new integrated technique, however, is that when a search is applied, the data encoded in the *x-position* and *y-position* properties is no longer valid. I.e., the positions of the marks in the map no longer indicate the geographic positions of the houses. One way to alleviate this problem is to animate the movement of objects from their original positions to their new positions. This will allow users to deduce useful information from the speed at which the objects are moving as well as provide users with context information about the object origin. In particular we would look for clusters of objects that are moving at relatively the same speed or outlier objects that are moving much faster than the other objects initially around them.

We can further extend the hybrid technique described in this section by allowing users to add several lines of reference that have different constraints attached to them (Figure A-13-left). In this case, each constraint line will apply a force onto the objects and the final position of the object would be the result of all these forces. Another variation of this technique is to use “constraint points” ((Figure A-13-right) instead of “constraint lines”. This would simplify the visual representation to some degree and allow us to put more constraints into the display. When we want to put in multiple constraint points it becomes very important *where* we place these points so that we can derive useful information from the resulting object positions. For example Figure A-14 shows an effective placement of three constraint points. Objects that are in the middle of the display are the ones that pass all three constraints, objects that are on the “in-between” points (indicated by 2’s in Figure A-14) pass two of the three constraints, and objects that are at the vertices pass only one constraint. As it turns out, the display shown in Figure A-14 is very similar to the InfoCrystal system [*Spoerri*, *1993*]; however, we arrived at the same design from a very different starting-point.

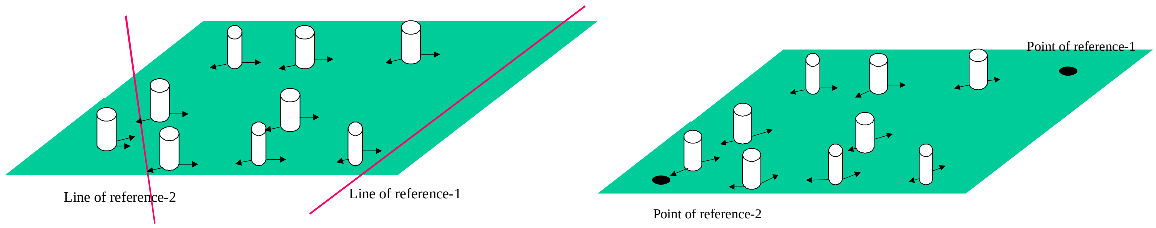


Figure A-13: Extended HomeFinder + SDM distance technique

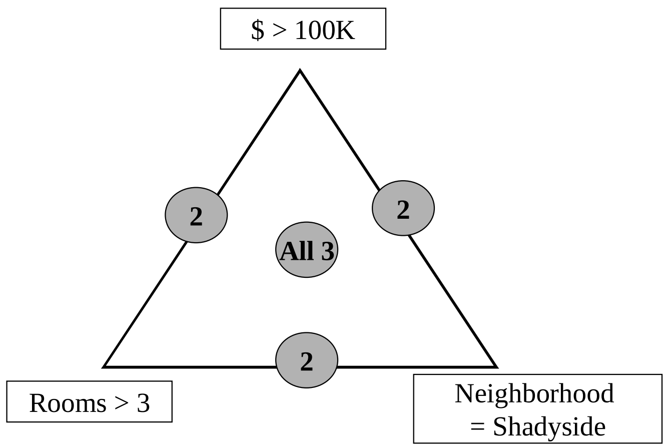


Figure A-14: Effective way of placing 3 constraint points

This section presents two detailed examples of how we can combine existing visualization techniques to form new and sometimes surprisingly novel behaviors that push the envelope of visualization technique design. Even in those cases where the combined results do not appear to have any clear use, we learn the strengths and weaknesses (new technique classes) and improve our ability to design future techniques.

A-3 Control Functions

One class of visualization functions that we did not consider in this chapter is *control functions*. Control functions regulate the flow of execution within a visualization method so that we may easily repeat operators, or choose between multiple different alternative functions. We did not include them in our description because the current visualization techniques we considered and the initial techniques we plan to automatically build with our design system do not require such functions. Future expansion of our design system however will profit significantly from the use of control functions. In this section we discuss some useful control functions and show how they may be integrated into our framework.

Sometimes it is necessary to repeat a set of object definition and transformation functions several times. Rather than having to declare the same specification over and over again, we can use *control operators*, to regulate the flow of execution of the function set. For example suppose we wanted to divide up a set of house concepts into three groups based on house price, and then color each set differently. In order to do this we could use the specification in Figure A-15 to divide up the concepts into houses that cost: 1) >= 100k and < 200k, 2) >= 200k and < 300k, and 3) >= 300k and < 400k. This can be achieved by repeating a pair of functional description functions and a color graphical transform three times; once for each house set.

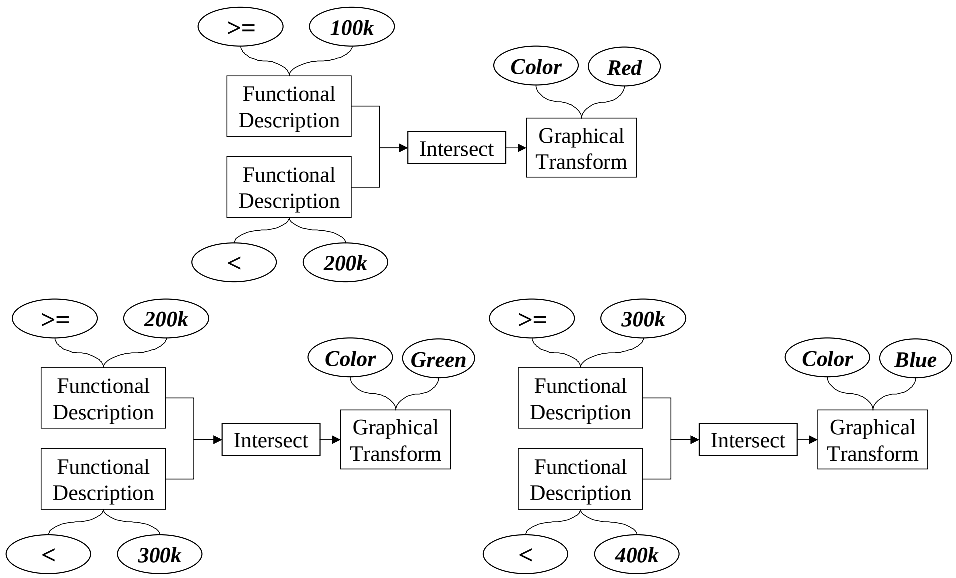


Figure A-15: Dividing up house concepts into 3 groups based on *selling_price*

A more efficient way to specify this task is to use a control operator to repeat the functional description functions and the color graphical transform so that we only need to specify them once. In Figure A-16 we use the *foreach* control operator on sets of threshold values and color values. The *foreach operator* is used to repeat a sequence of object definition and transformation functions for each member of a given set of elements. In this case the >=, <, and color graphical transform functions are repeated three times, once for each of the input arguments provided to the functions.

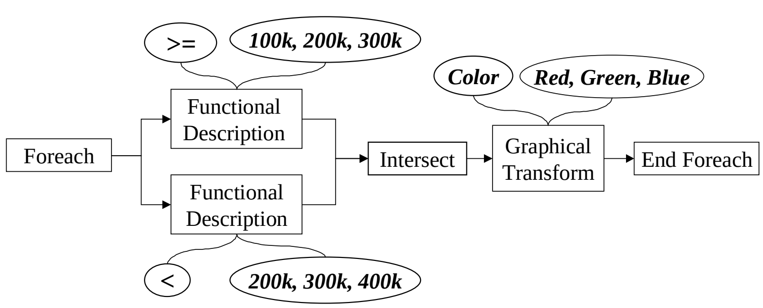


Figure A-16: Dividing up house concepts into 3 groups using the *foreach* control operator

Control operators also allow us to repeat a set of functions an indeterminate number of times, something that cannot be done through regular specification. For example, one interesting means of selecting objects is to divide up the object set into multiple partitions, where each partition contains values over particular ranges. This selection method is essentially a set of related threshold functions. We can achieve this behavior by first calculating the threshold values for each partition and then *P-C-composing*[2] that with a set of threshold functions. In the partition example shown in Figure A-17 we are calculating equi-distant partitions and then creating each partition with a pair of threshold functions. In this example the *loop* control operator is used to repeat the pair of threshold functions *n* times, where *n* is the number of partitions desired. Because we are using the *loop* control operator, we need not determine the number of partitions required during specification, and we can easily alter the number of partitions generated at any time without having to change the specification.

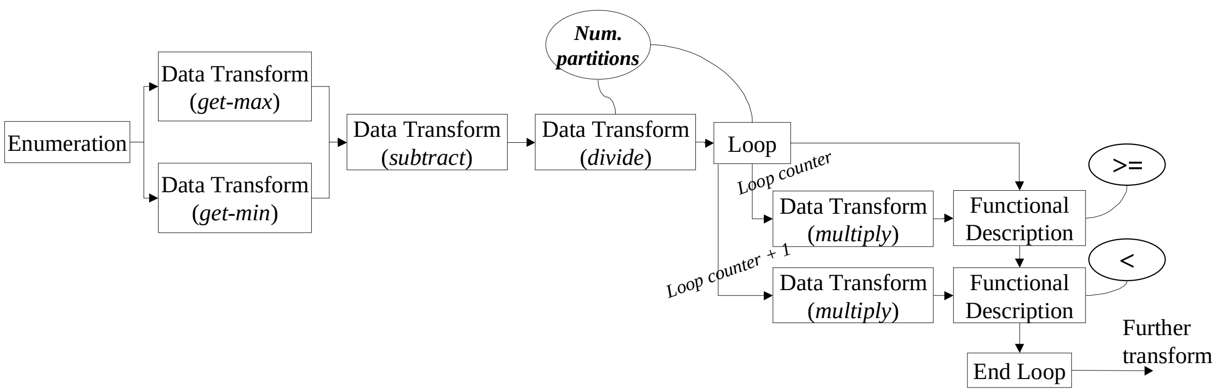


Figure A-17: Partition selection

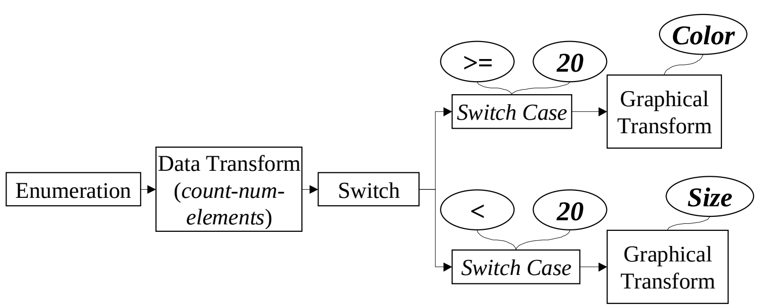


Figure A-18: Using the switch control operator to channel the execution flow

[2] *PC-composition* combines two primitive techniques by piping the outputs produced by one technique into the input slots of the other.

Finally there is the *switch* control operator that is used to choose among multiple different branches of execution based on a specified condition. For example suppose we want to highlight a set of user chosen house concepts differently depending on the number of concepts chosen. One way to do this is to use the *switch* control operator to channel the execution of the technique through different graphical transforms based on the size of the selected object set. In Figure A-18 we use the *switch* operator so that if the selected set has < 20 objects then its elements will be *color* highlighted, and if it has >= 20 objects then its elements will be enlarged (i.e. highlighted through a change in *size*).