Final version published as Mohiuddin, Arefin; Ashtari, Narges; Woodbury, Robert. Directly Interactive Design Gallery Systems: Interaction Terms and Concepts. In Ji-Huyn Lee (ed) *Computational Studies on Cultural Variation and Heredity*. Singapore: Springer, 2017.

Directly Interactive Design Gallery Systems: Interaction Terms and Concepts

Arefin Mohiuddin[1], Narges Ashtari[1], and Robert Woodbury[1]

[1]Simon Fraser University, Surrey, British Columbia, Canada
{amohiudd, nashtari, robw}@sfu.ca

Abstract. A human-computer interface interposes objects between a person and the underlying representation with which the person interacts. Previously, we introduced two interaction objects, alternatives and their collections in an interactive design gallery. We revisit the terms, refining their definitions, and introduce the explicit notion of a "view" to accommodate multiple references to the same alternative or collection in an interface. We outline fundamental interactions over alternative and collection views. Finally, we outline a special type of collection called the *Parallel Coordinate View*.

Keywords: Design exploration, design alternatives, computational models, computer aided design, design tools.

1 Introduction

In computational design, the case for design alternatives is established, and need not be repeated here. What is not widely understood is the need for a complete and effective suite of direct interactions with design alternatives. Instead, the literature largely reports techniques and systems for presenting to a user the results of some automated generative process. Yet, as Bradner et al. [1] point out, "Professionals reported that the computed optimum was often used as the starting point for design exploration, not the end product." We aim to remedy this situation by devising and evaluating a suite of direct interactions with design alternatives. We meet these aims through the iterative design and evaluation of prototype systems that represent multiple alternatives. For both intellectual and practical reasons, we build these prototypes on top of existing commercial parametric modeling systems. First, parametric systems are mature and increasingly used in practice–they are the best type of system for gaining research participants and external impact. Second, parametric systems yield a particularly simple and useful abstraction for representing alternatives [2], allowing clean separation of modelers from what we call design galleries for supporting alternatives. We have devised a general, flexible system architecture that enables us to rapidly prototype new interaction concepts.

2 Alternative Views

We revisit the definition of an alternative provided in [2] reproduced nearly verbatim (correcting only the capitalization of variables) herein.

1. An *alternative* is a selection and abstraction of nodes from a parametric model such that the nodes of the alternative each correspond to a property of a model node.

2. An alternative may contain both graph-independent and graph-dependent properties of the model it abstracts. The graph-independent properties enable alternatives to be edited; the graph-dependent properties enable performance to be assessed by evaluations tools, which are commonly available within a parametric modeler.

3. The operation *apply* assigns values from an alternative *f* to a model *m*. Specifically, it assigns the property values from the *f* only to properties that, in *m*, are graph-independent.

This led to overloading of the term "alternative" as a *logical alternative* (a subset of model nodes) and its interface counterpart, the *interface alternative* (a logical alternative plus a parametric model). It introduced two problems. First, it is verbose and its terms are not memorable. Second, it failed to separate underlying symbolic models from human-computer interfaces to them. Here we resolve the first problem by introducing the term "aspect" to replace "alternative," and using "alternative" to describe the former "interface alternative." The second problem we address through the term "view." In Model-View-Controller (MVC) frameworks, a common software architecture pattern for graphical user interfaces [4], a *view* is a representation of underlying information, and multiple views of the same information are possible. This is also consistent with information visualization principles where multiple visual representations of data should be available to support users with different tasks and requirements [5,6]. Our definition thus becomes the following.

1. A *model aspect* (or just *aspect*) is a selection and abstraction of nodes from a parametric model such that the nodes of the aspect each correspond to a property of a model node.

2. An aspect may contain both graph-independent and graph-dependent properties of the model it abstracts. The graph-independent properties enable aspects to be edited; the graph-dependent properties enable performance to be assessed by evaluations tools, which are commonly available within a parametric modeler.

3. The operation *apply* assigns values from an aspect *p* to a model *m* to produce an alternative *a*. Specifically, it assigns the property values from *p* only to properties that, in *m*, are graph-independent.

Using this new definition, since an *alternative* is a representation of the underlying aspect and parametric model, we adopt the term *alternative view*, and introduce it as a more nuanced concept.

In the Design Gallery implementation, an aspect and a model are stored in the alternative data structure, and an *alternative view* visually represents the alternative and is the primary interface object with which users interact. There may exist, as alternative views, multiple references to an alternative across the Design Gallery, and each may or may not be a different visual representation or *view*.

p = aspect
m = parametric model

An *alternative* = ${}_{j}a = \langle p,m \rangle$, where

j indexes alternatives, i.e., in a gallery G (galleries contain alternatives), ${}_{j}a \in G$ is the j^{th} alternative contained in G.

We introduce the concept *view*, denotes by *v,* and specialize is to alternatives.

alternative view = ${}_{j}v^{a}_{i}$, where

a denotes *alternative*, that it is of type *alternative,* as contrasted with an interface view of type *collection*.

j indexes alternatives, i.e., in a gallery G, j refers to the j^{th} alternative.

i is a *view index*. If, in a gallery G, multiple alternative views ${}_{j}v^{a}_{i}$ of ${}_{j}a$ were to exist, i denotes the i^{th} alternative view of the j^{th} alternative contained in the gallery. To elucidate, in the Design Gallery [3] , ${}_{j}a$ exists as views ${}_{j}v^{a}_{0}$ glyph with thumbnail image, ${}_{j}v^{a}_{1}$ larger thumbnail image, and ${}_{j}v^{a}_{2}$ thumbnail and text, and would be denoted as ${}_{j}v^{a}_{1}$, ${}_{j}v^{a}_{2}$, *and* ${}_{j}v^{a}_{3}$ respectively.

Note that these view kinds (thumbnail, large thumbnail, and thumbnail and text) are not exhaustive, and as our research progresses, we propose and design novel kinds of alternative views.

The model defaults to the one from which the alternative was initially abstracted, but is not restricted to it. Therefore, alternatives can be created by tuples selected from collections P of aspects and M of parametric models. This is consistent with the notions described in [3], i.e., an aspect can be applied to any model, even if the model changes. Such applications produce effects to the extent at node names correspond across models. *Fault tolerance* in the modeler prevents system crashes, and *partiality* allows aspects to be applied to models in which node mappings are either (or both) incomplete or extraneous.

This notation also gives rise to a novel way of generating alternatives within the Design Gallery. Sheikholeslami [9] describes the *Cartesian product* of values stored in an alternative or in a subjective node, whereas [7] extends this to products of parametric models. The Cartesian product $A = P \times M$, where P is a collection of aspects,

and M a collection of parametric models, is a collection A of all possible applications of $p \in P$ to $m \in M$.

3 Collection Views

Many desired operations act over multiple objects [3], that is *collections* of alternative views. A collection is

$$C = \{v^a\}.$$

Collections are mutually non-exclusive and collectively exhaustive. Mutual non-exclusion means that views to the same alternative can exist across multiple collections, that is, collections C and D can each contain a ${}_jv^a{}_i$ with the same j value. Collective exhaustion means that every alternative has an alternative view in some collection. We guarantee collective exhaustion by providing a *universal collection* containing one view of each alternative in a gallery. Collections are sets with respect to the alternatives underlying a gallery, that is, a collection can contain at most one alternative view of a given alternative. A *collection* is denoted as

$C = \{{}_0v^a{}_i, {}_1v^a{}_j, \ldots\ldots, {}_nv^a{}_k\}$, where
$\{0,\ldots,n\}$ indexes alternatives, and
i,j,k index their respective alternative views across all collections in the gallery.

This means that, alternative views of a single alternative may exist in multiple collections as a similar or dissimilar representation or *view*, but no collection contains more than one reference to the same ${}_ja$ even if the *view* is different. Effectively, in ${}_jv^a{}_{i,}$ i indexes across all alternative views of j present in the gallery.

Interacting with *collections* of *alternative views* necessitates that there be different representations of these *collections*, again in line with information visualization research [5,6]. An example may be visualizing relations between elements of collections. Thus, we introduce the term *collection views*. A collection view *i*s a visual representation of a collection C--collection views form the primary interface objects by which users interact with collections. Following the convention set previously, we denote a collection view as

${}_jv^c{}_i$, where

j indexes collections, i.e., in a gallery G, j refers to the j^{th} collection.

i is a *view index*. If, in a gallery G, multiple collections ${}_jC$ were to exist, i denotes the i^{th} collection view of the j^{th} collection contained in the gallery.

c indexes collection view types, an extensible set of visualization types provided by the gallery. Currently, the gallery defines three collection view types:
c = c.general. The general or default view of a collection, where *alternative views*

appear clustered in a user directed layout (Figure 1.right),
c = c.parallel. A *parallel coordinate controller* view (Figure 1.left).
c = c.pareto. Multiple Pareto graphs in which properties of an alternative may be plotted against each other (Figure 2),

These would be denoted as

$$v_1^{\text{c.general}}, v_2^{\text{c.pareto}}, v_3^{\text{c.parallel}}$$

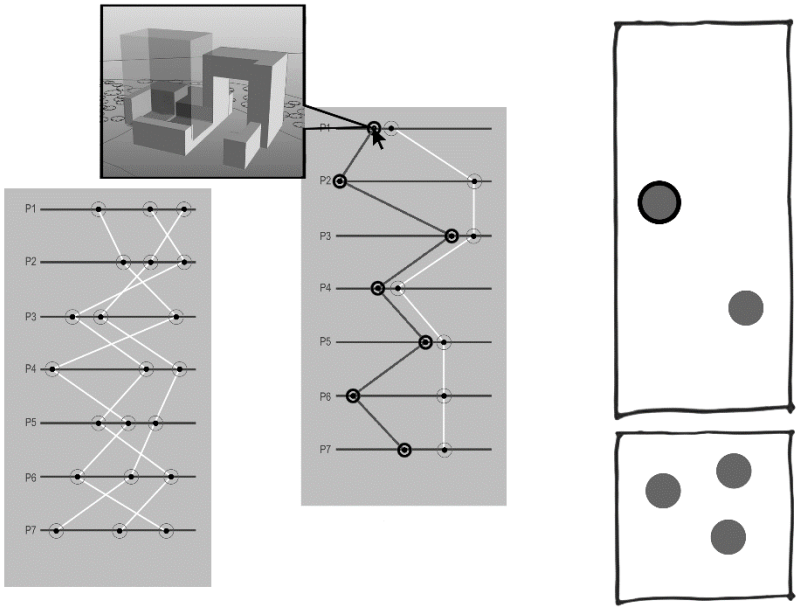
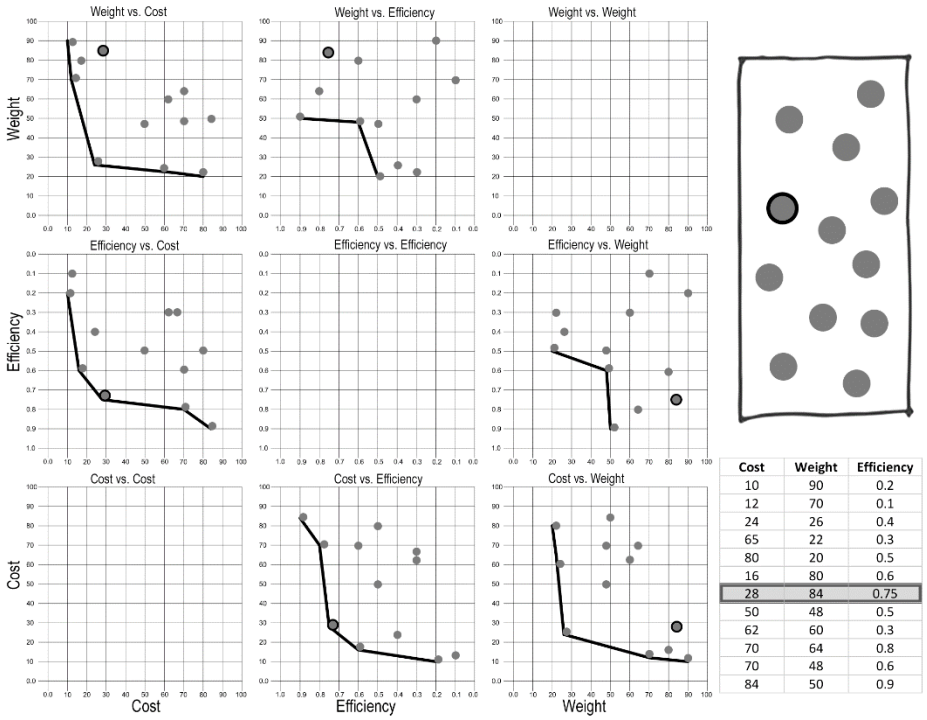


Fig. 1. **(left)** Parallel coordinate view controller (left) showing alternatives and handles, details on demand show 3D view, and (right) brushes corresponding general alternative view as a glyph in general collection.

4 Interactions

The fundamental interaction with interface objects is selection. The two primary interface object types v^a and v^c give rise to two distinct selection sets s^a and s^c. Users should be able to select discretely or continuously, multiple v^a and/or v^c preceding any operations that may follow. Selecting an object adds it to its type's *selection set*. A selection set is also a collection (though this is not strictly true for s^c), except that it is ephemeral in nature. We describe here a core set of interactions upon which more detailed interactions depend.



Cost	Weight	Efficiency
10	90	0.2
12	70	0.1
24	26	0.4
65	22	0.3
80	20	0.5
16	80	0.6
28	84	0.75
50	48	0.5
62	60	0.3
70	64	0.8
70	48	0.6
84	50	0.9

Fig. 2. A collection view as multiple pareto views showing representative performance criteria evaluated against each other. One ${}_j v^a$ is highlighted across all views.

The primary acts of interaction using a mouse pointer in the gallery are "click" and "drag". A modifier, which may be one or more key-presses, different mouse buttons, click sequences, or any combination thereof, is applied to specialize these. Tables 1, 2 & 3 show primary interactions on the two principal types of target objects, the modifier used, and the result. For modifiers, we indicate both an integer identifying the modifier and our current design decision binding a particular action to the modifier. The following interactions we explicitly define hold true for $v_1^{\text{c.general}}$; while being theoretically valid for $v_2^{\text{c.pareto}}$ and $v_3^{\text{c.parallel}}$, they have case specific implications that are a work in progress.

The selection convention found in most interfaces is exactly the reverse, i.e., unmodified clicks select single objects, and deletes existing selection sets. Since our tasks almost always involves working with multiples, we eschew this convention in favor of its reverse. This is less error prone, as large selection sets can be lost by unintentional clicks, frustrating users. Accidental deselection is harder, because actions required are more deliberate. Explanatory tool-tips or introductory screens during adaptation will improve the learnability of this new convention.

Table 1. Interaction on null objects with modifiers and results. At the time of writing, we bind modifier 1 to SHIFT, modifier 2 to the ALT key and modifier 3 to CTRL.

Target object	Interaction	Modifier	Result
Null	Click	Null	Null
Null	Click	1	*Clears* s^c, the collection selection set.
Null	Drag	Null	Adds to the selection set s^c all v^c within the boundary of the selection rectangle. This implies that selection is cumulative.
Null	Drag	1	*Subtracts* all v^c within the boundary of the selection rectangle from s^c. Deselection is also cumulative.
Any	Double-click	Null	Creates a v^c of a new collection C from s^c. if s^c is empty, create a new empty collection. This effectively merges all collections in s^c. No effect on selected v^c. The original s^c persists, but now includes the new v^c.
Null	Click	1 & 2	*Clears* s^a, the alternative selection set.
Null	Drag	2	Adds to the selection set s^a all v^a within the boundary of the selection rectangle. This implies that selection is cumulative.
Null	Drag	1 & 2	*Subtracts* all v^a within the boundary of the selection rectangle from s^a. Deselection is also cumulative.
Any	Double-click	2	Creates a new ${}_j a$ and corresponding ${}_j v_0^c$ for each v^a in s^a. No effect on selected v^a. The original s^a persists.

Table 2. Interactions on alternative views v^a with modifiers and results.

Target object	Interaction	Modifier	Result
v^a	Click	Null	*Adds* clicked v^a to s^a. (Figure 3)
v^a	Click	1	*Subtracts* target from s^a.
v^a	Click	1 & 2	*Clears* s^a, and *adds* the clicked v^a to s^a.
v^a or s^a	Drag from *source to target* v^c	Null	*Adds* a new v^a to the collection C of target v^c. Does not remove the target v^a from source v^c. Clear s^a and add the new v^a to s^a. (Figure 4)
v^a or s^a	Drag within v^c	Null	Changes screen position of v^a in v^c if the type of v^c so allows, otherwise NULL.
v^a or s^a	Drag from *source to target* v^c	1	*Adds* the v^a to the collection C of target v^c. Removes it from source v^c. Clear s^a and add the new v^a to s^a. (Figure 5)
v^a or s^a	Drag within v^c	1	Null.

Table 3. Interactions on collection views v^c with modifiers and results. For brevity and clarity, we omit cases where a source v^c and target v^c refer to the same C.

Target object	Interaction	Modifier	Result
v^c	Click	Null	*Adds* clicked v^c to s^c. (Figure 6)
v^c	Click	1	*Clears* s^c, and *adds* the clicked v^c to s^c.
v^c or s^c	Drag	Null	Changes screen position of v^c in gallery. (Figure 7)
v^c or s^c	Drag	2	*Add* all v^a from the collection C of source v^c to that of target v^c. The display of v^a in v^c depends on the type of v^c. No effect on selected v^c. (Figure 8)
v^c or s^c	Drag	3	Makes a new v^c of the collection at the end-of-drag screen location.

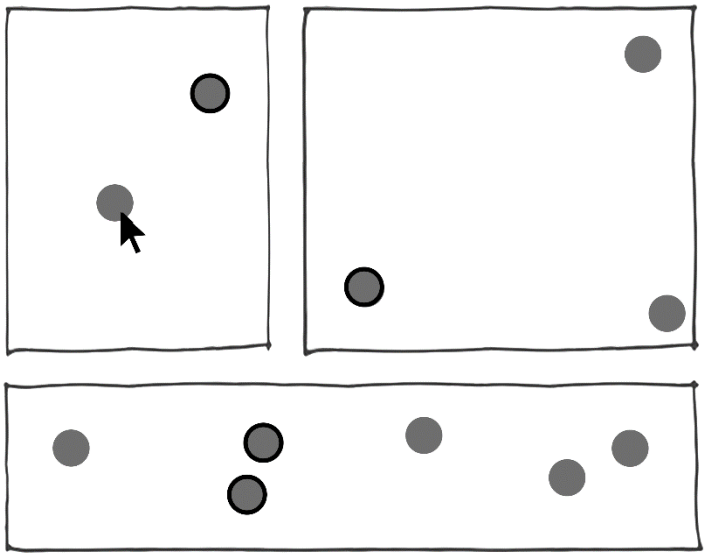


Fig. 3. Clicked v^a will be added to s^a.

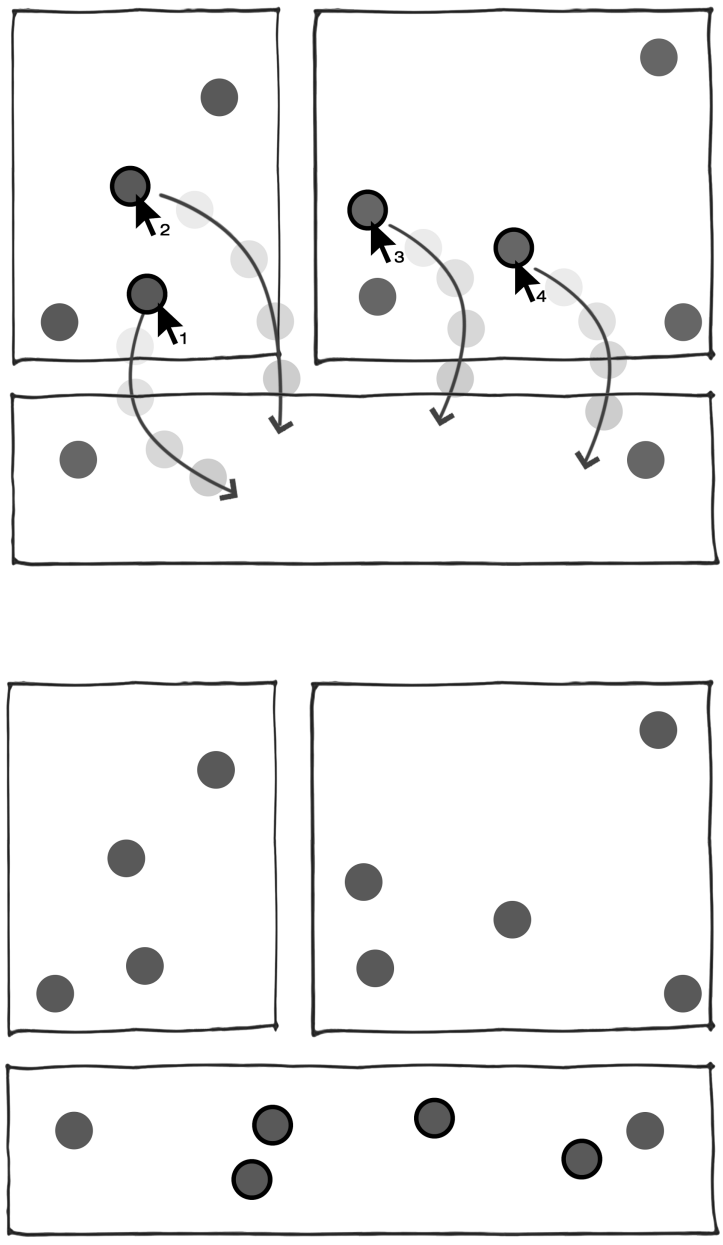


Fig. 4. Dragging from source ${}_{i}v^{c}$ to target ${}_{j}v^{c}$ (top) adds v^{a} to target ${}_{j}v^{c}$ (bottom) without deleting from source ${}_{i}v^{c}$.

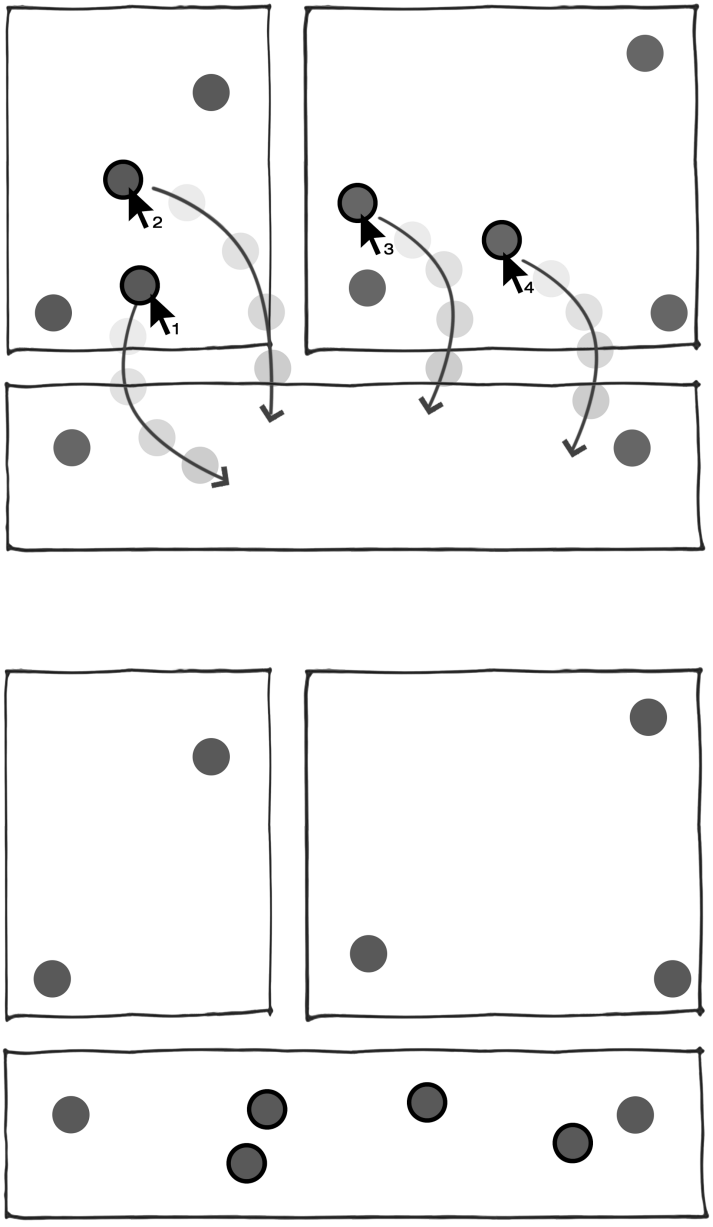


Fig. 5. Dragging from source to target vc (top) adds va to the target collection and removes the dragged v^a from the source collection (bottom).

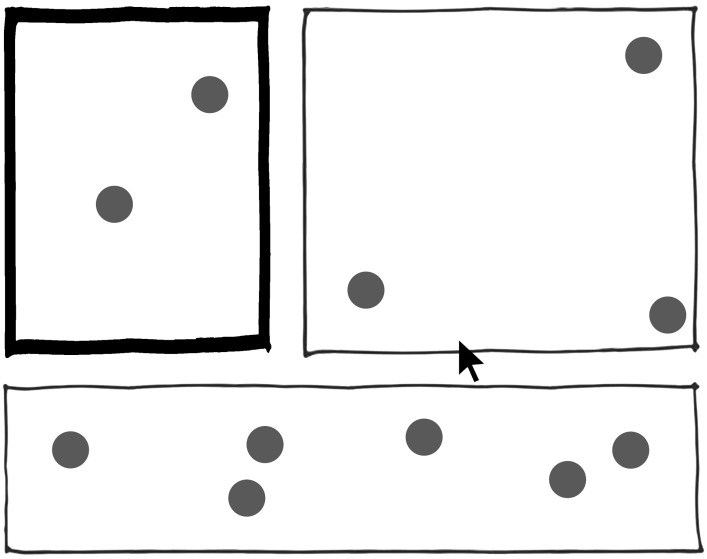


Fig. 6. Clicking on a v^c adds it to the s^c .

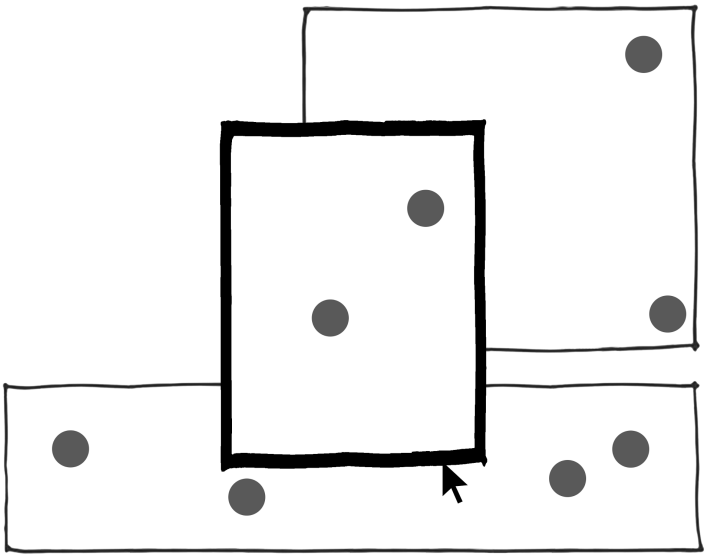


Fig. 7. Dragging a v^c on another v^c superposes the selected v^c without affecting the v^c located underneath.

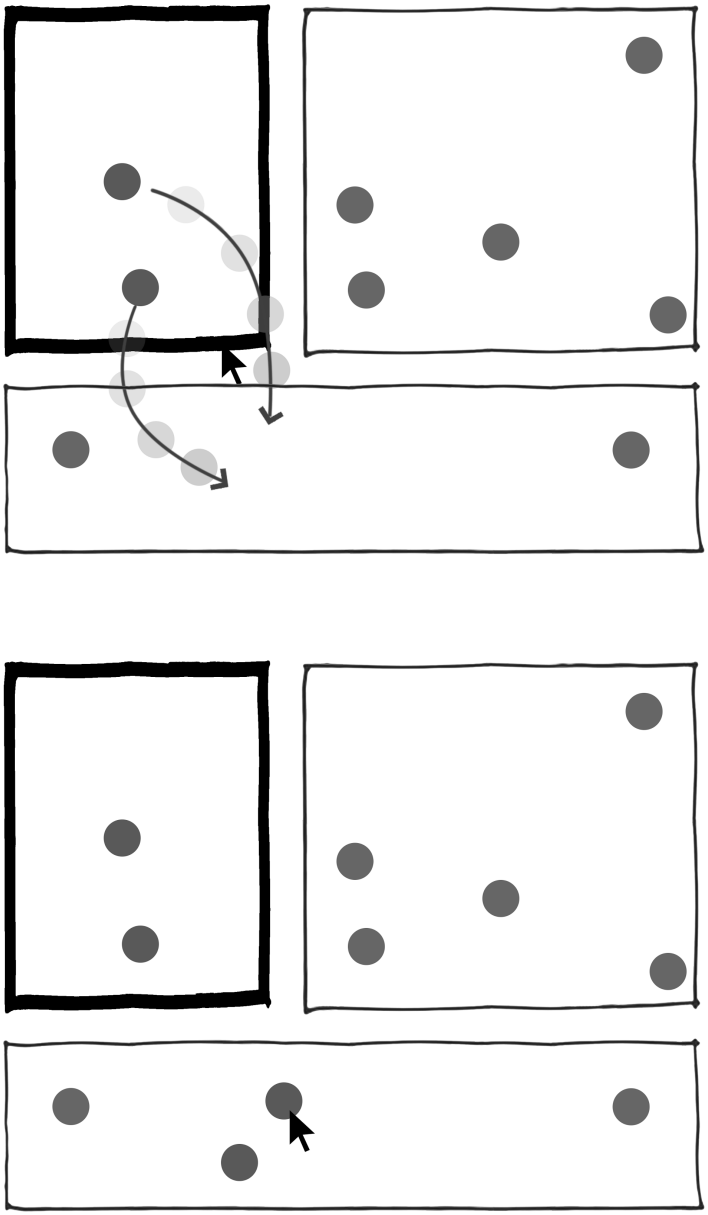


Fig. 8. Dragging a v^c with Modifier 2 (top) on another v^c adds the v^a from the dragged collection to the target collection based on the type of the destination type (bottom).

5 Introducing the Parallel Coordinate View-Controller

5.1 Motivation

Woodbury et al. [8] propose the idea of combining designers' past decisions in new arrangements, by which designers may be able to discover new meaningful alternatives. They coin the term *design hysteresis* to describe operators that perform such recombination. They call the set of states already visited the *explicit* design space, and the set all possible states the *implicit* design space. They introduce the concept of *hysterical space* (after *hysteresis,* the lagged entry of an effect into a system) to describe the result of operations that use states in the explicit space to access those in the implicit space.

Sheikholeslami [9] describes a specific case of such an operator, the *Cartesian product* of graph independent properties of an invariant parametric model, and coins the term *hysterical state* to describe states derived out of this recombination and, after Woodbury and Burrow, calls the set of such states the *hysterical space* (Figure 9). In this context, the explicit design space is the set of states visited by the designer, and the implicit design space is the set of states achievable by exhausting all parameter values, which in the case of continuous parameters, is indenumerably infinite. The hysterical space is the Cartesian product of all parameter settings recorded by the designer.

To interact with the hysterical space, Sheikholeslami proposes the *Dialer* (Figure 10), comprising concentric rings, where each ring represents one parameter and the divisions on the ring correspond to the recorded values of that parameter. The outermost ring contains the Cartesian product recorded parameters. Each ring has a slider with an adjustable size that selects the values on the rings. By moving and resizing the sliders, one can select the desired values for highlighting the corresponding items in the outermost ring (hysterical space). The shortcomings of this interface are that it is not scalable, as the number of divisions increases with number of recorded variations; and the number of concentric rings increases with parameters. The hysterical space also increases exponentially, thus making the dialer unreadable.

We use the Cartesian product in the Design Gallery as the expand operator. Interactions in the gallery closely follow the interaction model described by Sheikholeslami. Graph-independent node properties of every aspect in the gallery corresponds to the recorded parameters in the explicit design space. In our Grasshopper™ implementation, users may choose to create a "pool" by recording states, or pick individual parameter values from alternatives in the gallery as candidates for the Cartesian product. In both cases, the exhaustive Cartesian product is calculated in the modeler, and a dialer (Figure 11) lets users browse through members of the product in rapid serial visual presentation. In our study [2], we found that participants extensively used the expand operator and to rapidly generate variations and scan through them using the dialer before selecting alternatives of value. This strategy was dominant across all users, and participants verbally confirmed preference for this form of interaction. However, participants used only a few "seed" alternatives to generate relatively fewer variations--foresight tells us that, as the Cartesian product expands exponentially, linear scan through results by rapid serial visual presentation will become a less meaningful way of interacting with alternatives.

This brings to us the design challenge of visualizing, interacting with, and navigating large hysterical spaces (e.g., as generated by a Cartesian product), in the specific case described by Sheikholeslami, as well as the larger problem of design space exploration. Hysterical space is multi-dimensional data in nature, and we turn to *parallel coordinates*, a widely used and effective way of visualizing multi-dimensional data [10]. A vast body of literature exists on efficient and enhanced use of parallel coordinates for data visualization and exploratory data analysis, as well as widespread use of it in academia and industry.

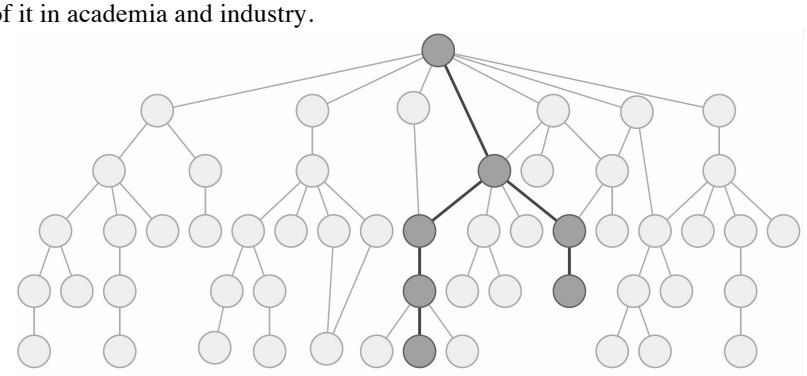


Fig. 9. Brushed paths depicts the explicit design space that a designer passes to reach a solution. Unbrushed paths depict other possible paths not taken, i.e., implicit design space. Note that the implicit paths would not be represented in any interface as to do so would make them explicit in some sense. Image credit Sheikholeslami [9].

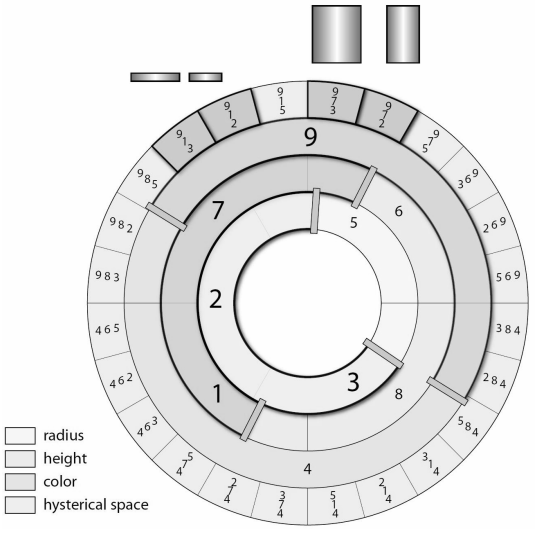


Fig. 10. A Dialer for a simple parametric model of a cylinder with radius, height, and color as input. Outer ring represents hysterical space, Cartesian product of recorded inputs. Image

credit Sheikholeslami [9].

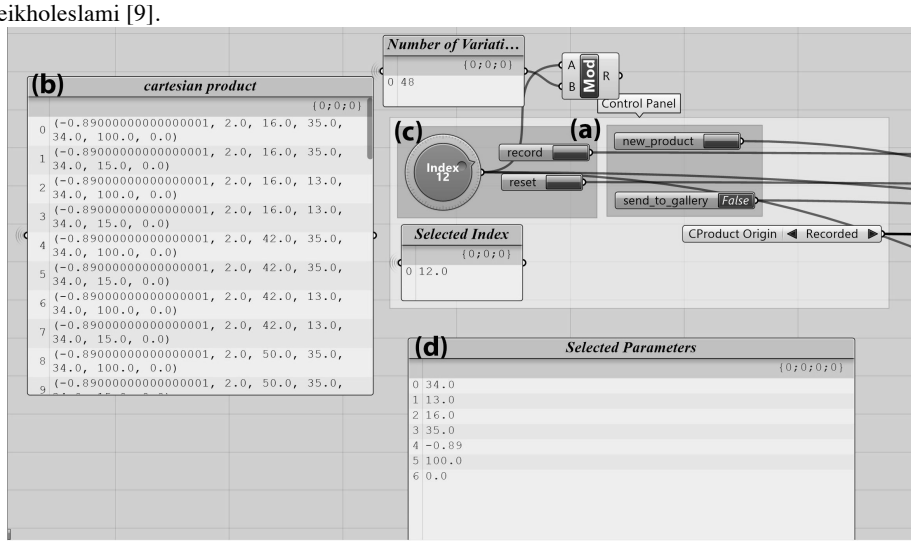


Fig. 11. Cartesian Product dialer in Grasshopper™ implementation of Design Gallery (a) record button to record parameters (b) Cartesian product or hysterical space as vectors (c) dialer, and (d) parameters of member at selected index.

5.2 Design Overview

The choice of using parallel coordinates follows nearly directly from parametric modeling interfaces. Extant parametric modeling tools use horizontal sliders as the most common interface for varying input parameters. It is not uncommon to find complex graphs with numerous slider nodes aligned horizontally. In information visualization, parallel coordinates are typically vertically oriented, where the up direction signifies an increasing value and vice versa. In our Parallel coordinate view-controller (henceforth abbreviated as PCVC), we change this to a horizontal alignment to match sliders, maintaining familiarity in the design discipline.

A PCVC is a special case of a v^c. Graph independent and dependent nodes from all $m \in M \in C$ form each axis in the PCVC. The property values of the $p \in P \in C$ in the v^c are plotted on the axes, marked by a circular "handle". In the case of ordinal properties, they increase from left to right, and default bounds are set by minima and maxima found by querying the property. In the case of nominal properties, discrete points are formed whose order may be changed. A v^a is therefore represented by a line running through the handles. Selecting a line selects the v^a, and selecting a handle selects the individual property value for that v^a only. Selecting an axis selects every value for all v^a that intersect that axis. Multiple lines may be selected independent of multiple handles across axes. Handles may overlap, and on selection attempts, a pick parade is proposed. Handle selection sets up an *create* or *edit* operation described later.

The ordering of the axes themselves are arbitrary, or in the order they were encountered from when the aspect was created, however it is typical in implementations of parallel coordinates to allow re-ordering. This is necessary because parallel

coordinates transform the search of multivariate relations in the dataset to a pattern recognition problem, and such rearrangements help in gaining insights[10]. However, independent and dependent nodes remain in separate groups for clarity.

Selection of two or more lines and computing the Cartesian product will yield all the possible lines through handles of those lines. As new alternatives are added to the v^c, the axes are automatically populated. There are many methods of maintaining and enhancing readability of dense parallel coordinates in the literature that may be applied when the view is densely populated. Brushing and filtering are some common operations. Therefore, the PCVC avoids at least some of the scalability issues that plague Sheikholeslami's *Dialer*. A parallel coordinate view may be brought up by toggling any v^c, or it may be viewed side-by-side in combination with other view types of v^c. As an example of a $v^{c.general}$ interaction valid in $v^{c.parallel}$, dragging lines ($_jv^a$ or s^a) from $v^{c.parallel}$ to any target v^c will add $_jv^a$ or s^a in the default view. Dragging within $v^{c.parallel}$ results in a null operation.

5.3 Generative use

We propose that, in addition to visualizing values, each axis also behave as an input interface, hence the suffix "controller". Moving a handle will effect changes in the corresponding graph independent properties in the aspect *p*. As a consequence, a new alternative *a'* will be computed by *applying* this modified aspect *p'* to *m*. Multiple handles can be changed, and this will create multiple new *a'* in parallel. (Figure 12) If the choice is not to retain the original *p*, then this operation will be an *edit*. Drawing a line through the graph independent axes by dragging the mouse will create a *p* (Figure 12) by creating values at the intersection points of the line and axes, while the user has freedom to choose the associated *m* for the <*p,m*> tuple to create a new *a*. Thus, the model may vary, unlike Sheikholeslami's proposal. This new *a* appears in the controller's v^c. We propose that moving handles for graph dependent properties trigger multiple goal seeking operations which result in new alternatives that meet the new performance criteria. Currently, we have implemented parallel generation/editing of new alternatives using graph independent properties. We employ a server-client architecture. A request from the gallery sends the new <*p,m*> to a remote server running an instance of the same modeler, which applies *p* to *m* to create a new *a*. Sufficient computing power will allow continuous and realtime update of the 3D view.

5.3 Summary

The PCVC is our current focus of work. It is evident from the short overview that there are numerous possibilities for rich interactions to be designed and evaluated. We are possibly the first to propose that a visualization tool be also used as a generative tool. Information visualization principles go hand in hand with creativity support guidelines. A large literature in each area poses problems that need to be addressed and solutions that can be applied. However, as with the Design Gallery system, our immediate goal is to describe the most fundamental interactions.

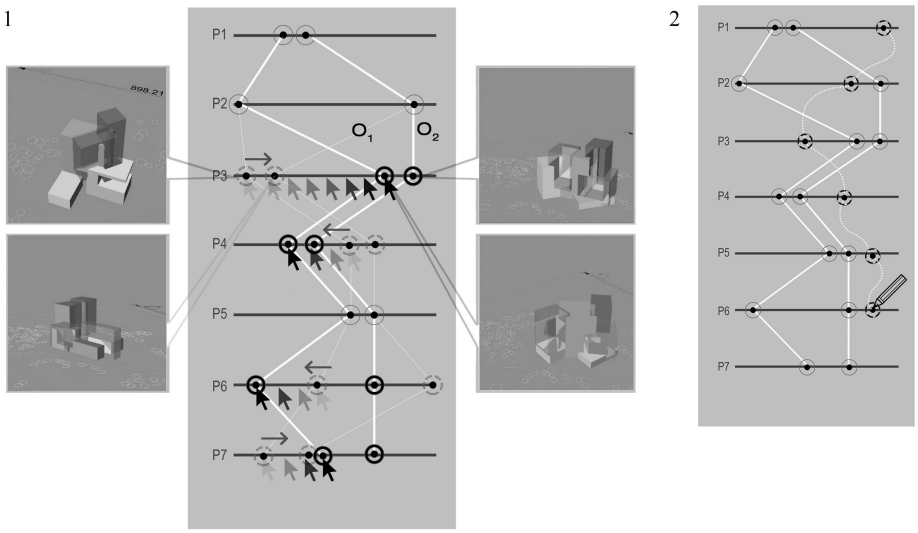


Fig. 12. (1) Dragging handles computes new alternatives. (2) Drawing a line through the axes creates a new aspect.

6 Future work

There exist a finite set of available interaction controls (e.g., CTRL, ESC, ALT, right- and left-mouse click). Despite finiteness, it is evident that a sufficiently rich set of interactions arise, and devising a coherent and consistent encoding commands using these controls is a major design challenge. For example, Modifier 1 and Modifier 2 in the above may both be mapped to CTRL. This is our current work and the subject of a future paper.

The gallery system is also designed for high-resolution large displays [3]. This too adds to the design challenge on every interaction aspect, for example, selecting distant objects in the gallery. It also raises the question whether the traditional mouse and keyboard is adequate for such a challenge [11,12] given the easy availability of advanced pointing devices and touch enabled displays. This too is the subject of our future research.

References

1. Bradner, E., Iorio, F. & Davis, M. Parameters Tell the Design Story: Ideation and Abstraction in Design Optimization in Proceedings of the Symposium on Simulation for Architecture & Urban Design (Society for Computer Simulation International, Tampa, Florida, 2014), 26:1–26:8.
2. Mohiuddin, A., Woodbury, R., Cichy, M., Mueller, V. & Ashtari, N. A Design Gallery System in ACADIA 2017: Disciplines & Disruption 414- 425 (ACADIA, Boston, MA, 2017).
3. Woodbury, R., Mohiuddin, A., Cichy, M. & Mueller, V. Interactive design galleries: A general approach to interacting with design alternatives. Design Studies 52. Parametric Design Thinking, 40 –72 (2017).
4. Gamma, E., Helm, R., Johnson, R. & Vlissides, J. Design Patterns: Elements of

Reusable Object-Oriented Software (Addison-Wesley Professional, 1995).
5. Tory, M. & Moller, T. Human factors in visualization research. IEEE transactions on visualization and computer graphics 10, 72–84 (2004).
6. Munzner, T. Visualization analysis and design (CRC press, 2014).
7. Zaman, L., Stuerzlinger, W., Neugebauer, C., Woodbury, R., Elkhaldi, M., Shireen, N. & Terry, M. GEM-NI: A System for Creating and Managing Alternatives In Generative Design in Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (ACM, Seoul, Republic of Korea, 2015), 1201–1210.
8. Woodbury, R., Datta, S. & Burrow, A. Erasure in Design Space Exploration in Artificial Intelligence in Design 2000 (Kluwer Academic Publishers, Worcester, Massachusetts, 2000), 521–544.
9. Woodbury, R. Elements of Parametric Design With contributions from Brady Peters, Onur Yüce Gün and Mehdi Sheikholeslami (Taylor and Francis, 2010)
10. Inselberg, A. *Parallel coordinates: Visual multidimensional geometry and its applications* (Springer, New York, 2009).
11. Robertson, G., Czerwinski, M., Baudisch, P., Meyers, B., Robbins, D., Smith, G., and Tan, D. (2005). The large-display user experience. IEEE Computer Graphics and Applications *25*, 44–51.
12. Czerwinski, M., Robertson, G., Meyers, B., Smith, G., Robbins, D., and Tan, D. (2006). Large Display Research Overview. In CHI '06 Extended Abstracts on Human Factors in Computing Systems, (New York, NY, USA: ACM), pp. 69–74.