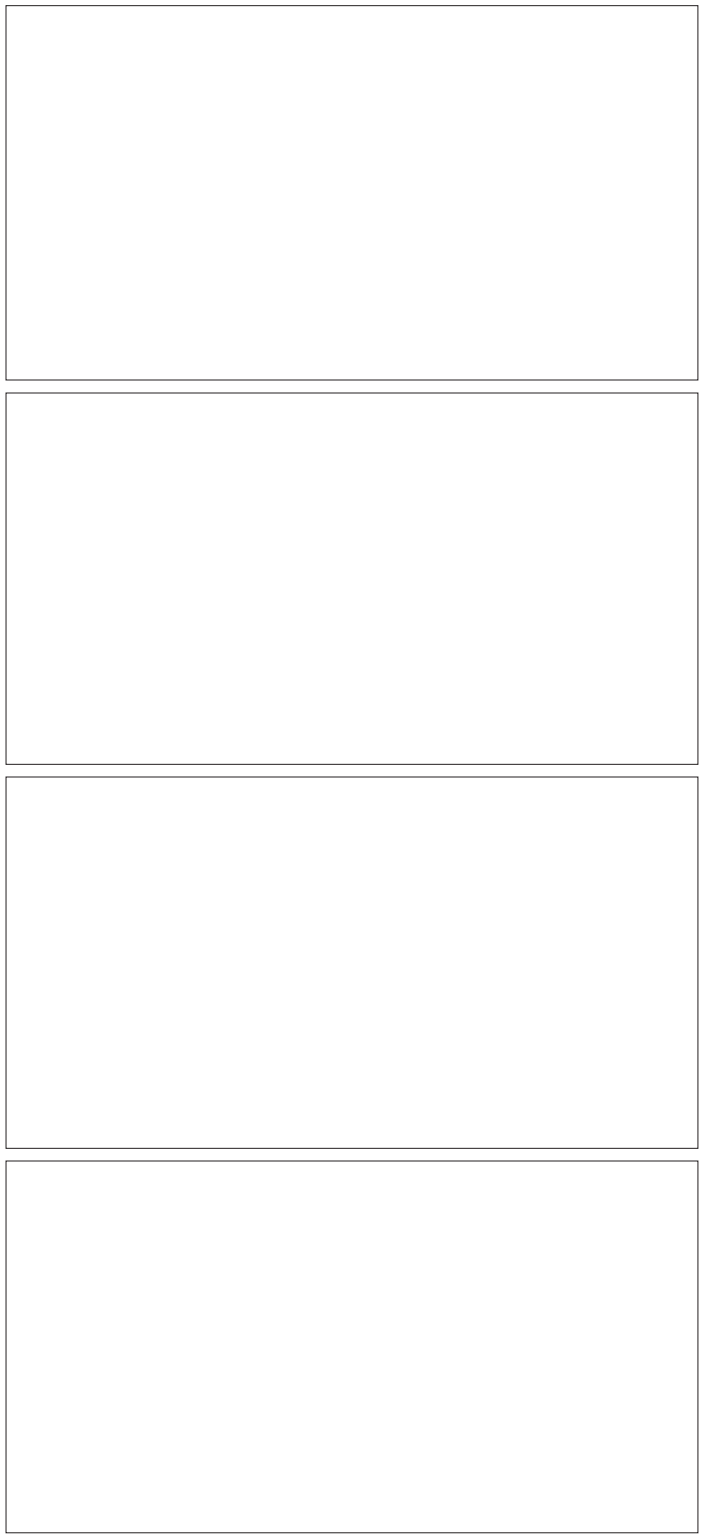
Undo Reinterpreted

Wilson Chang, Rob Woodbury
Simon Fraser University



Abstract

The class of operations known as “undo” has proven to be a valuable addition to most professional work tools. In practice though, its use is frustrating: undo often undoes too much. Its essential informal semantics are that it returns the user to a prior state by recapitulating all intervening states. Why not give the user greater control over which aspects of a design to undo? An alternative is to seek to reuse prior work in any logically-coherent pattern—user input is a precious commodity. The area of generative systems provides insights in a search for alternatives to undo, in particular that prior user and system actions can be changed and reused in new contexts. We contingently introduce a concept we label as design promotions to describe system designs that demonstrate a tight coupling between interactive authorship and system-led generation, that treat past user actions as valuable intentional statements, and that treat alternative user choices as first-class objects of concern. In practice these three properties emphasize reuse. We briefly survey the current state of undo-like operations and potential candidates for implementing design promotions strategies. Through examples, we demonstrate approaches to realizing undo-like operations over specific representations, especially that of constructive solid geometry.

Undo Reinterpreted

Wilson Chang, Rob Woodbury, Simon Fraser University

1 Introduction

Research in generative systems shows how alternative designs can be generated through the application of formal devices such as ontologies (Woodbury et al. 1999) and rules (Heisserman 1994). Realistic generative systems must work in vast solution spaces that defeat any attempts at completely automatic search. This is why the most convincing demonstrations of generative systems involve the user as an integral part of the process. There have been several intriguing interfaces developed for generative systems, and some of these are exemplars of the crafts of user-interface design and programming. There is, however, little theory to guide future development in how users can interact with generative capabilities.

Extant CAD systems present surprisingly similar problems. They provide the user with an often large array of design modification operators and relatively weak mechanisms for recovery of past states of a representation. Reuse is typically limited to insertion of previously saved files and a symbol instancing facility such as the symbols of Form Z (Form Z 2003b), the Xref of AutoCAD (AutoCAD 2003b), or the Reference File of MicroStation (MicroStation 2003a). Some advanced systems such as Bentley's CustomObjects (in the AEC industry) or CATIA (largely in the mechanical engineering industry) support the user in building parametric design representations, in which design variables are interrelated through relations (typically equations and inequations) and the system computes settings for the variables that are consistent with the relations. Such systems support reuse of work through quick recomputation of design variables after user changes, but remain largely silent on both higher levels of system assistance and on the capture and presentation of choices to the designer.

A key, perhaps the key, concept here is intent. Under an interpretation of designer action as intent, the undo/redo operators of conventional CAD systems can be seen as a means to return to prior intended states of a design. They are the main (perhaps the only) explicit record of design intent stored in a conventional system. Undo/redo was a powerful addition to most interactive authoring systems in the late 1980s and early 1990s. In its most frequently supported form it allows a user to undo to prior states of a design and to redo the undone operators provided that no new operations interceded between the undo and redo steps. In programming concept it is a pair of simple LIFO stacks. When a state is popped from the undo stack, it is pushed onto the redo stack. The redo stack is emptied on every new user operation. Of course, this simple mechanism can and has been implemented in a variety of ways, for instance through undoable operators and journaling mechanisms (Berlage 1994). Undo was lauded at the time, and most users of any sophistication rely on it today. It is clearly inadequate as a means to recover past work, as conversations with any group of users will reveal. It is typical that an operation to be undone lies well down the undo stack. Deletion of previous valuable work is often necessary, including related sub-concepts that the user would want to retain in future versions. Users are forced to delete valuable data in the undo process, making it necessary to reconstruct information back into the design. Sophisticated users develop idioms of use around undo. For example, they use the clipboard as a way of storing data that will be undone but that needs to be restored; they use a manual process of using existing data to build a scaffold for constructing new data prior to emulating undo through the manual deletion of data; and they manually patch existing data to approximate past data. These idioms are workarounds for a situation that seems obvious: undo as it is presently implemented in most systems is overly constraining. It often deletes more of a user's prior intent than is logically necessary.

Parametric design systems support designer intent in a fundamentally different way than do conventional CAD systems. Their main strategy is shared with a very familiar class of software: the spreadsheet. A parametric model is a template for a design expressed as a data structure containing variables, a set of relations amongst the variables that must be met in any instance of the template, and a set of assignments of values to the variables that comprise an instance of the design. Users of parametric modeling systems develop all three aspects of the model, and this can be a technically complex task. The payoff for all of this work is that a model can be varied. Changing one (or some) of the instance variables of a model causes the system to rewrite the other variables to be consistent with the expressed relations. (Some parametric modelers, for instance, most spreadsheets and CustomObjects, implement a model of computational flow from independent to dependent variables. Others, for instance, those of Gross (1990) and Borning and Freeman-Benson (1998), allow change to virtually any instance variable.) The intent here is the set of relations between variables in a model. In concept, parametric systems add a second level of undo to a system. One can (or should be able to) undo changes to the relations independent of changes to the values of the design variables. Both though recapitulate the problems encountered in conventional CAD systems: it often

seems that more needs to be undone than is logically necessary. This creates extra work for the user.

Clearly, promoting designs to more complex states, demoting them to previous states, and reusing aspects of design work are all important to users. We contingently advance a concept we call design promotions as a combination of three features. It labels system designs in which generative tools are directly linked to interactive authorship and act to both promote and demote designs between less and more organized forms. It views past user actions as statements of intent, which may be edited to alter designs and reused in new contexts. Promotion/demotion and reuse imply that a fine-grained notion of choice and presentation of alternative choices be a part of the interface.

Design promotions embodies the idea that users' choices and actions are, in themselves, valuable objects and that users should be free to edit, reuse, and reflect on past choices and freely make new choices. The concept stresses the user—demonstration of its utility will be at the level of user tasks. Any number of well-known symbolic devices for supporting design change could well be used to implement design promotions. For example, the rules or type hierarchies of generative systems could be seen as a substrate for generalizing past experience. Extant systems are exemplars of choice in the moment—they typically provide many different ways to model designs, ranging in grain from global to local details of a representation. They are, however, primitive in their support of past operations, mostly limiting themselves to simple forms of undo. This paper begins with demonstrating the undo devices of current systems and shows how the information usually captured in undo stacks can be used in more powerful ways. For example, it is generally the case that the order of operations specified by a user provides more information than is necessary to model an object—it is a serialization of a process that has significant logical parallelism. Users should be free to revise models by undoing and redoing operations in any logically feasible order. Thus, in undoing or redoing, users may arrive at designs that they have never before made. Extant CAD systems provide hints at such capability. For example, in 3D Studio Max, primitive objects, such as box, allow undo of parametric modifiers in any order.

Undo though is only a synecdoche of a broader class of objects in CAD systems that would be amenable to reinterpretation under the concept of design promotions. In principle, any data structure created through the action of a user can be interpreted as a (partial) statement of the user's intent, and it may well be worthwhile to see if there are ways to rewrite the data structure to produce other models. If such other models contain less data than the original, that is, something is deleted from them, what the user will see will be related to undo in that it removes information, yet, unlike undo does not follow the strict logic of recapitulating the user's sequence of operations. A consequence of such a strategy is that the user will be able to "undo" operations to "return" to places never before encountered.
To implement a design promotions system it seems clear that at least two things are needed: a representation over which fine-grained change can be recorded and edited; and user interfaces for working with such fine-grained representations; neither are well-evidenced in the literature. Fine-grainedness is a necessary property for a system that does design promotions—the intent is to be able to alter literally every conceivable designer action. There is a literature on the related idea of selective undo, for example, "UNDO and REDO Operations for Solid Modeling" (Toria et al. 1986).

Woodbury et al. (1999) have developed a representation for design space exploration that appears to be suitable for implementing design promotions. This representation is an augmentation of typed feature structures (Carpenter 1994). It displays properties of partiality and intentionality that give it the requisite fine-grained structure.

Partiality is important because it allows us to have a representation of something without having all properties of the thing to hand. For example, we can know of the existence of a line without knowing the precise locations of its endpoints. We can know it has a colour without knowing its colour. A representation displaying this property comes with a way of incrementally changing properties, one by one. An intentional representation has the property that two identical representations do not necessarily model the same object. This allows us to make incremental decisions about how objects relate—we can, for instance, decide at any time that columns in two separate structural systems are, in fact, the same column.

Woodbury et al. (2000) show that there exists a natural operator over typed feature structures called hysterical undo. This operation allows a user to unwind operations done on a feature structure representation in every feasible order. The term hysterical refers not to an emotional state, but to the scientific sense of the word as designating the lagged entry of an effect into a system. In this case the cause is a statement of designer intent. Its effect as a design enters the system only when a particular subset of prior declared intentions is chosen. Over this one representation system, there appears to be a suitable algorithm for design promotions.

Revisiting undo and redo operations reveals how we might actually give a user control over such an algorithm. A first idea is to tie the notion of selection to that of undo. In such a scheme, undo would apply to the user-selected set of objects. With each selection, the system generates probable state transformations. Hysterical undo proposes possible demoting paths to a less organized state. The new state opens up opportunities for design reuse and promotion again into a more desirable design. Undo and redo operations provide specific and partial realization of design promotions. The undo mechanism provides a safety net for user to freely navigate the design space. The redo mechanism provides the liberty of reusing any feasible concepts. It is not clear that this is the only, or even a workable scheme for user interaction with a system supporting design promotions. An investigation of current undo/redo mechanisms reveals limitations

and hints at richer interpretations of undo/redo operations for design promotions.

2 Extant examples of UNDO

Most applications incorporate variations of a linear undo model. In such a model, all executed commands are stored in a history list. To undo a command pop it out from the history list and push it into a redo list (Berlage 1994). The current linear undo models restrict the flow of design by documenting only a single working path. It limits the exploration nature of the design task and hinders reuse of past effort.

2.1 AutoCAD 2000i

AutoCAD (AutoCAD 2003a) incorporates a simple linear undo model. It records a history list and offers the user the possibility to redo one operation. Undoing sequentially will step backwards in the history list one operation at time, effectively returning to prior states. The history list can be switched off, or contain only one undo step to save computing resources. Users can group several commands as a single operation in the history list. For grouping to work, the user is required to set BEGIN and END pairs of checkpoints. The grouping operation is not intuitive. Users often cannot foresee the need for such grouping utility until several commands have been executed. A MARK feature is provided. When BACK is called, the history list returns to the state where the last MARK was set.

2.2 Form Z 3.8

Form Z (Form Z 2003a) incorporates a linear undo/redo model. Undo returns the project to its state before the most recent operation. Undo may be executed sequentially until all operation items are popped from the history list. Redo cancels the immediately precedent undo. As with sequential undo, redo may also be executed sequentially. The ability to sequentially redo after undo is an advantage over AutoCAD's single redo function. Users are able to visit previous states and use the Redo-All function to return to the state before the last sequential undo. Reset-Undo/Redo clears both history and redo lists to avoid overwhelming system memory. When a non-undo command is issued, the redo list is cleared and the new command is push into the history list. Clearing of redo list presents a problem since designers might need to return to previous designs during an undo/editing process.

2.3 Microsoft Word 2002

MS Word (Microsoft Word 2002 2003) provides a linear undo/redo model similar to the one in Form Z. User can undo sequentially as many actions as needed. The redo list contains all the actions that have been undone as long as no new command is issued. MS Word has an interface that allows the user to see both the history list and redo list. The visual representation of the history list provides clarity and predictability to the undo/redo process. One notable attribute is MS Word's extension to its redo function (Ctrl+Y). The redo function works as repeat function when there is no item in redo list.

2.4 Non-linear undo

3D Studio Max implements a limited form of selective undo, but only within the values of independent variables in its parametric primitive subsystem. It is one of a very small number of systems that implement a non-linear undo logic in any way. It demonstrates well that selective undoing of a parametric modifier at a lower part of modifier stack can result in forms never before encountered. In fact, the differences with prior forms can be dramatic indeed.

2.5 Scenarios in Microsoft Excel 2002

Some systems (e.g., spreadsheets, parametric modelers) model data dependencies as intentional acts (the user builds the dependencies with the expectation of their maintenance). Even in such systems, the maintenance of scenarios (prior settings of properties) is an issue. Note that MS Excel (Microsoft Excel 2002 2003) has a scenario system by which states of independent variable sets can be checkpointed. Excel's scenario tool is, in UI terms, quite removed from direct interaction with a spreadsheet.

3 Reinterpreting undo as design promotions

In this section we demonstrate two kinds of undo-like operations. Both treat prior user actions as serious statements of intent. Both support the reuse of user-created data structures to "return" a user to states never before seen. This makes them no longer undo operations. They are rather instances of a design promotions strategy—since they remove information we might well call them design demotions, but we will use the positive term promotions to describe both information adding and information removing operations.

3.1 Simple properties

The properties of objects exist or do not. Objects can have other objects as properties (all objects within a document can be considered properties of the document). Typically a property adopts all its sub-level properties as well. For example, selecting wood veneer as a surface property inherent the color, grain, reflection, and smoothness sub-properties of such surface type. Every change to an object introduces a new property of the object and undoes an old property. All old properties are potentially available to be reset. A selection of objects defines a set of old properties that may be reset. In the process, the generative mechanism proposes possible property combinations that the user might have missed. One simple example is choosing contrasting color pairs. Black and red is the first pair. Deep blue and bright orange is the second pair. An alternative combination might be black and bright orange. A user interface for such resetting would make old properties available based

on user selection of objects. For example, clicking on an object pops up a list of properties appropriate for resetting. Moving over a property item display attributes for the property. A simple click resets the object with the selected property.

The independent variables of a parametric system are essentially simple properties of a model. They can be freely set without respect to other properties. In the parametric case, the system is responsible for tracking the effects of such change. We propose a scenario setting tools for CustomObjects (MicroStation 2003b). CustomObjects framework by Bentley Systems supports an object-oriented and parametric approach to modeling complex designs. Properties are propagated automatically through relationships built into a model. Designers as model programmers build relationships between parts and explore designs interactively. The design freedom of the system is commendable, yet designers are not able to document favorable model states. The proposed scenario setting tools would record independent variable sets for fast retract of design states. A user interface of the scenario manager would allow naming states for selected objects. A mouse click of such selection displays past scenarios to which a user might return.

3.2 Promotions in Boolean expressions

Constructive Solid Geometry (CSG) uses binary trees (with unary transformation nodes) as the data structure to store Boolean operations for a solid. The Boolean operators over solid models define a language of solids in which each object is (at least implicitly) associated with the CSG tree of operations used to create the solid. Of course, there are multiple CSG trees defining every solid. The particular choice of modeling operations defines one for each created solid. Further, every CSG tree corresponds to an algebraic expression that can be reduced to equivalent expressions by equivalence operations. The particular expression actually used in a modeling sequence represents, in some way, the intention of the user in a design process. Figure 4 illustrates the associations of CSG tree to the combinations of primitive shapes to form a simple massing model for a building (for the sake of rhetorical clarity, it omits the unary transformation nodes, the purpose of which is to locate primitive objects in space).

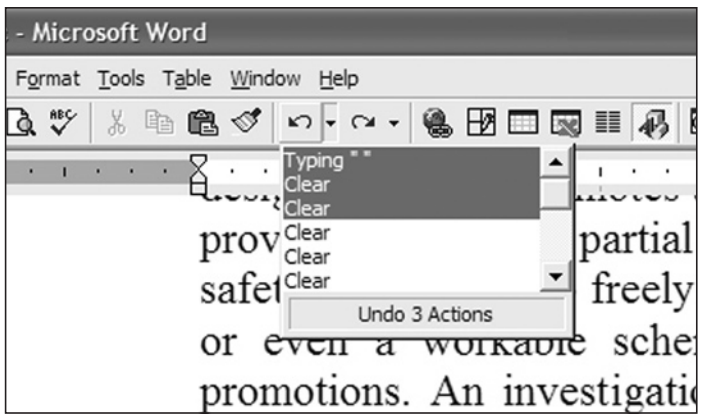
One definition of a design promotion in such a representation is that an operation can be simply removed from the expression. The results are independent expressions, no longer joined by an operation. Any of the operations in a Boolean expression can be undone—the result, using policy of treating the first operand as primary, is always two Boolean expressions yielding well-formed objects. Figure 5 shows examples of deleting selected operators from a Boolean expression.

A second strategy for Boolean expressions is to remove a chosen solid. Since every solid in a CSG tree identifies a path to the root of the tree and every path consists of zero or more operations, such a move provides opportunities to suggest different alternatives. Figure 6 shows examples of primitive deletion. By traversing up from the primitive nodes at the leaves of the tree, alternative choices can be generated.

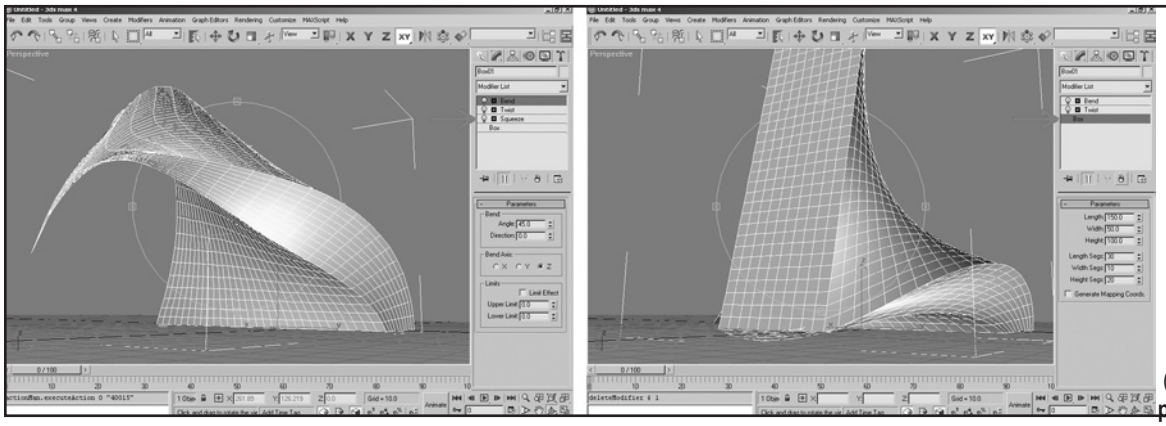
A user interface for undoing primitives can be structured around picking a face—every face comes from a primitive object, and every primitive defines a path of operations leading to the root of the CSG tree. Any such choice is a candidate to undo a given face. A choice of multiple faces produces a path from the join of the argument paths to the root of the CSG tree. Upon selection, the system would prompt for the available alternatives.

4 Conclusions

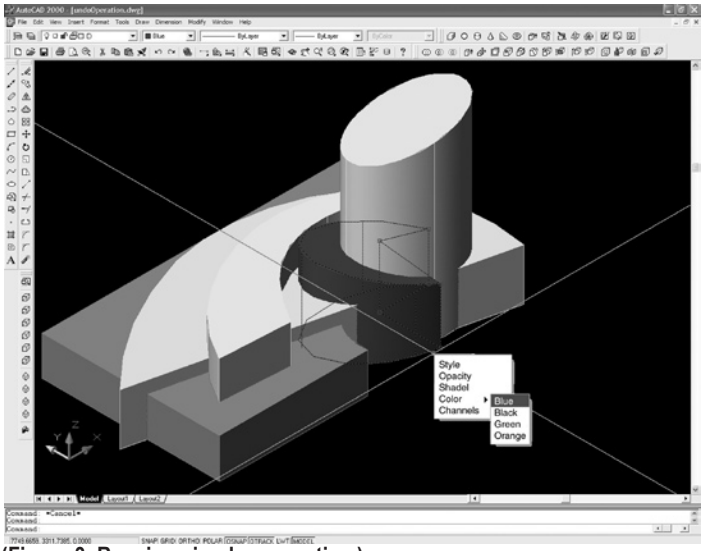
Undo remains a highly useful and widely implemented idea. Its utility here is to suggest more general forms of reuse of designer actions. In concept every representational structure created by a user embodies an aspect of intent. The system design problem is to bring such past intent to the surface for present consideration. Such a process is not, in general, deterministic, and so is complicated by requiring user choice. The examples presented in this paper aim at specific representations, while it is clear that commonality across representations is important if an interface idea is to be broadly applicable. Nonetheless, the awkwardness of contemporary undo makes worthwhile a search for more logically sensible alternatives.



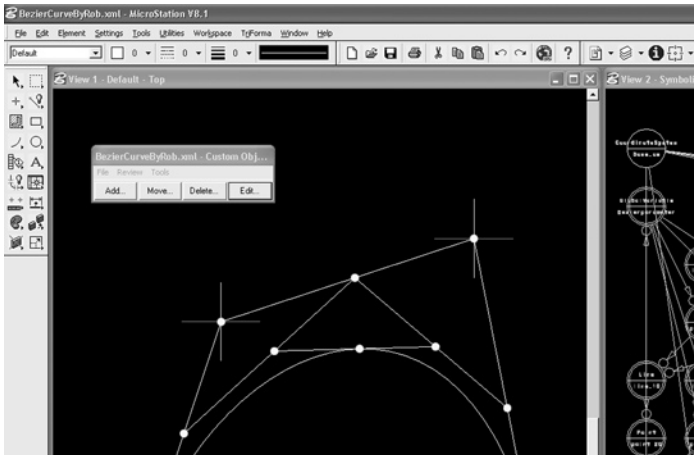
(Figure 1. Undo in Microsoft Word 2002.)



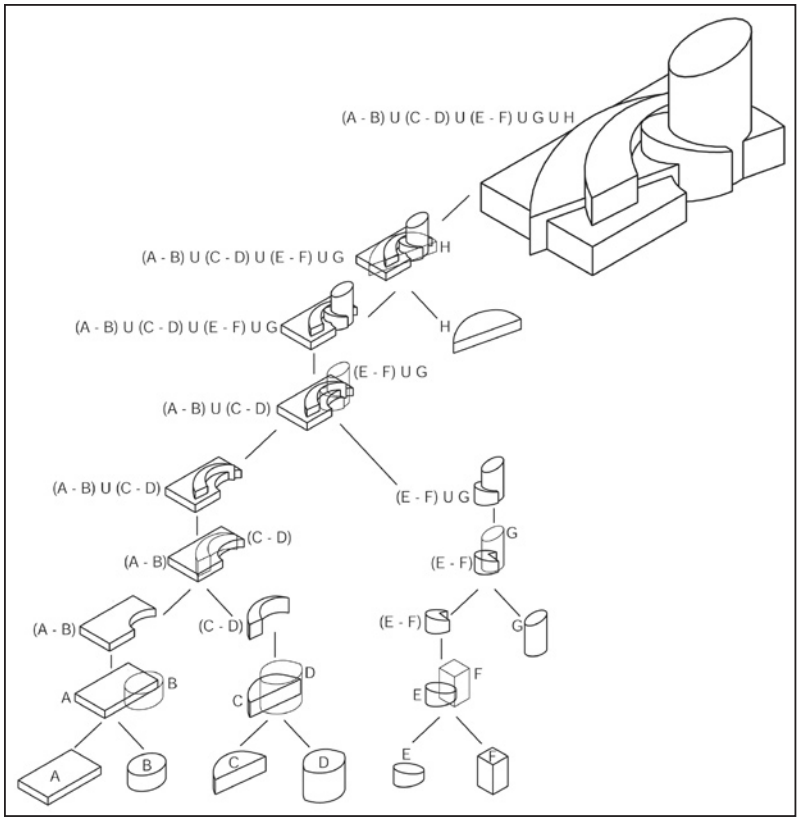
(Figure 2. Unexpected result of undoing a parametric modifier.)



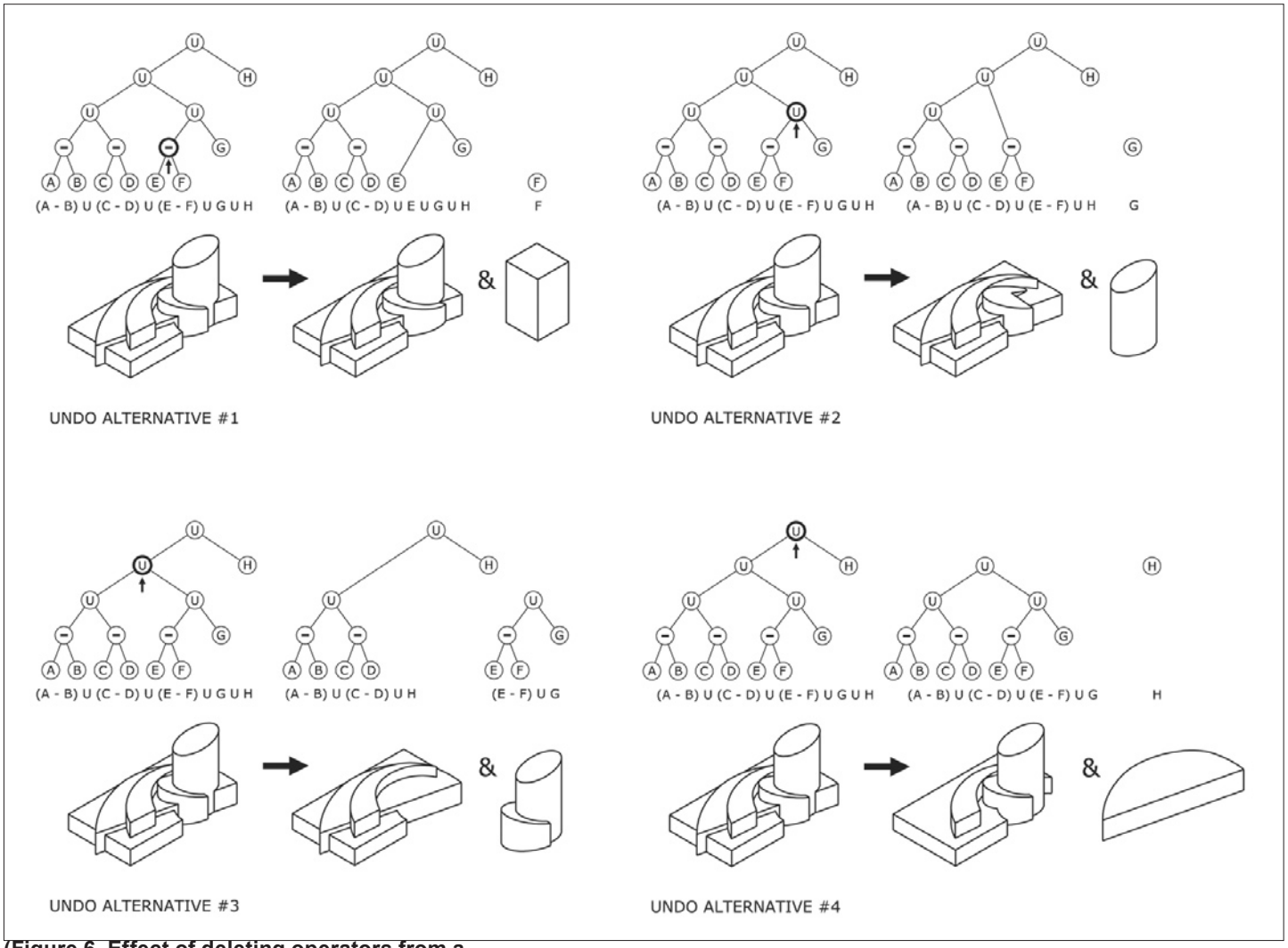
(Figure 3. Reusing simple properties.)



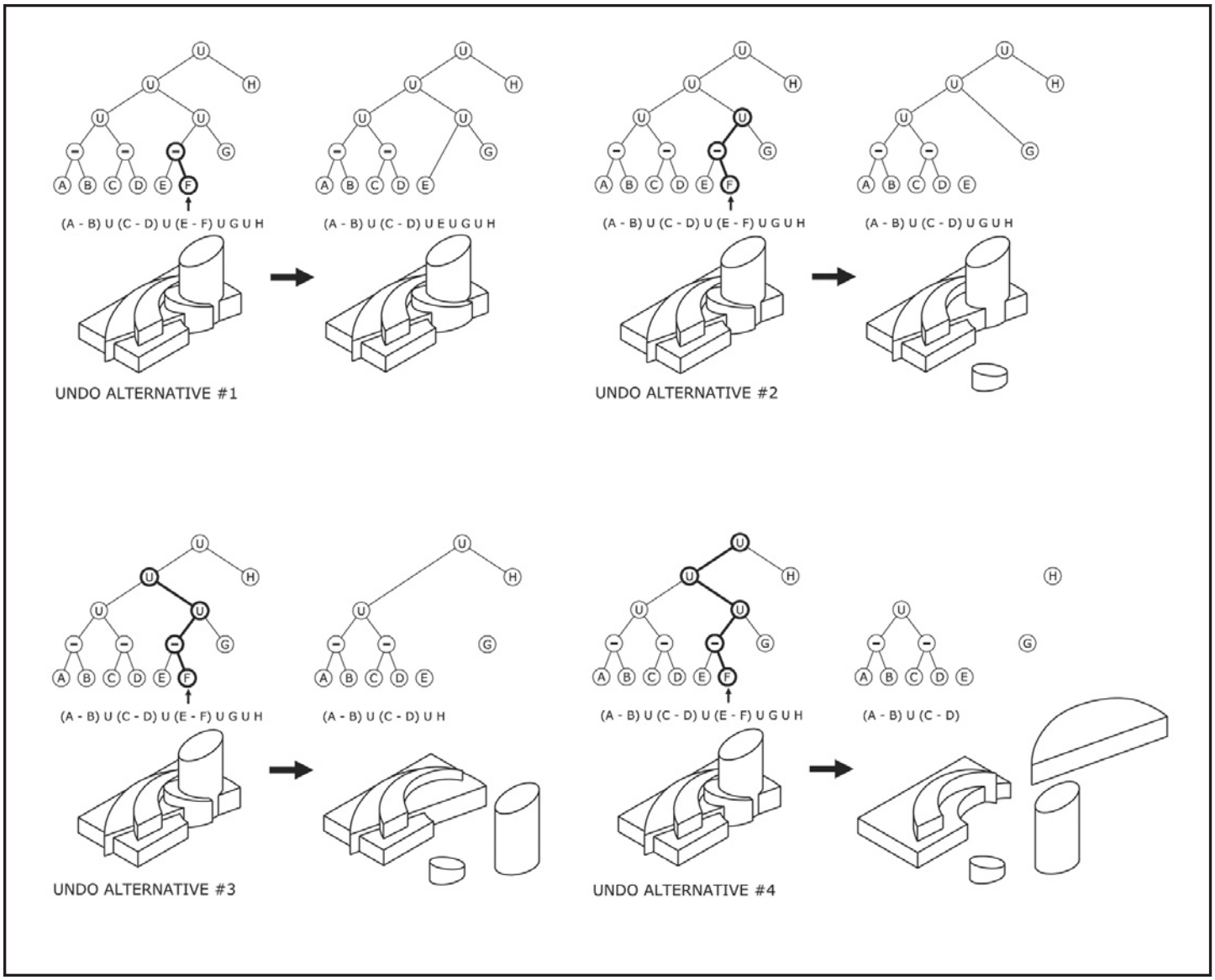
(Figure 4. Proposed Scenario Manager for CustomObjects.)



(Figure 5. CSG tree representation of a building massing.)



(Figure 6. Effect of deleting operators from a CSG representation of a building massing.)



(Figure 7. Effect of deleting primitives from a CSG representation of a building massing.)

References

AutoCAD, Found on the WWW at *http://www.ids-cad-software.co.uk/arch_con_AutoCAD2000i.shtml* on 16 May 2003.

AutoCAD Xref, Found on the WWW at *http://www.laep.ced.berkeley.edu/classes/la132/htdocs/x_2002/x-10/ex10.html* on 16 May 2003.

Borning, A., and B. Freeman-Benson. (1998). Ultraviolet: A constraint satisfaction algorithm for interactive graphics. In *CONSTRAINTS: An international journal*, Special Issue on Constraints, Graphics, and Visualization 3(1): 9-32. New York: Kluwer Academic Publishers.

Berlage, T. (1994). A selective undo mechanism for graphical user interfaces based on command objects. *ACM transactions on computer-human interaction*, vol. 1(3):269-294. New York: ACM Press.

Carpenter, B. (1992). *The logic of typed feature structures*. Cambridge: The Cambridge University Press.

Form Z, Found on the WWW at http://www.autodessys.com/ on 16 May 2003.

Form Z Symbols, Found on the WWW at *http://www.mmhk.com/e/products/autodessys/symbols.htm* on 16 May 2003.

Gross, M. (1990). Relational modeling: A basis for computer-assisted design. In *The electronic design studio: architectural knowledge and media in the computer era*. CAAD Futures '89 Conference Proceedings, 123-136. Cambridge

Heisserman, J. (1994). Generative geometric design. *IEEE* 37-45. New York: The Institute of Electrical and Electronic Engineers.

Microsoft Excel 2002, Found on the WWW at http://www.microsoft.com/office/excel/default.asp on 16 May 2003.

Microsoft Word 2002, Found on the WWW at http://www.microsoft.com/office/word/default.asp on 16 May 2003.

MicroStation Reference File, Found on the WWW at http://selectservices.bentley.com/technotes/technotes/8235.htm on 16 May 2003.

MicroStation CustomObjects, Found on the WWW at http://www.bentleyuser.org/LarsWord/larsMar2003.htm on 16 May 2003.

Toria, H., T. Satoh, K. Ueda, and H. Chiyokura. (1986). UNDO and REDO operations for solid modeling. In *IEEE computer graphics and applications* 6:35-42. New York: The Institute of Electrical and Electronic Engineers.

Woodbury, R., A. Burrow, S. Datta, and T-W. Chang. (1999). Typed feature structures and design space exploration. In *Artificial intelligence for engineering design, analysis and manufacturing (AIEDAM) special issue on generative system in design* 13:287-302.

Woodbury, R., S. Datta, and A. Burrow. (2000). Erasure in design space exploration. *Artificial intelligence in design 2000*, 521-544. Worcester: Key Centre for Design Computing.