Introduction: A New Agenda for Computer-Aided Design

William J. Mitchell

Harvard University Graduate School of Design

Design is the computation of shape information that is needed to guide fabrication or construction of an artifact. This information normally specifies artifact topology (connections of vertices, edges, surfaces, and closed volumes), dimensions, angles, and tolerances on dimensions and angles. There may also be associations of symbols with subshapes to specify material and other such properties.

The process of design takes different forms in different contexts, but the most usual computational operations are transformations (unary operations) and combinations (binary operations) of shapes in a two-dimensional drawing or a three-dimensional geometric model. An initial vocabulary of shapes, together with a repertoire of shape transformation and combination operators, establishes the shape algebra within which the computation takes place.

The computation terminates successfully when it can be shown that certain predicates are satisfied by a shape produced by recursively applying the transformation and combination operators to the initial vocabulary. These predicates are usually stated in symbolic (verbal or numerical) form. Thus determination of whether a predicate is satisfied usually involves producing a numerical or verbal interpretation of a drawing, then deriving inferences from this interpretation by applying rules or formulae.

This definition may seem provocatively reductionist and to leave little room for creativity. But I shall argue, in this paper, that taking a computational view of design can reveal precisely where creativity enters and why we intuitively take it to be characteristic of all but the most trivial design processes. In particular, I shall focus on the roles of ambiguity and discontinuity in shape interpretation, instability in the rules for carrying out shape computations, and nonmonotinicity in critical reasoning to determine whether or not a design proposal is complete and satisfactory. My examples will be taken from the sphere of architecture, but I shall provide a general theoretical treatment that applies to many other areas of design as well.

Describing and Manipulating Shape

A computer drafting system (or three-dimensional modeling system) models a shape algebra in essentially the same way that a four-function electronic calculator models the familiar algebra of real numbers. The calculator's display shows a number in the set of numbers that carries the number algebra, and the drafting system's display shows a shape in the set of shapes that carry the shape algebra. The calculator's keyboard provides a set of operators (addition, subtraction, multiplication, division) for manipulating numbers, and the drafting system's menu provides a set of operators for manipulating shapes (insertion, deletion, translation, rotation, and so on). The calculator is useful because we can employ numbers to represent balances in bank accounts, or areas of rooms, and we can then employ operations on numbers to represent operations on bank accounts or rooms. The drafting system is useful because we can use shapes to represent spaces and construction components, and we can then employ operations on shapes to represent operations on those entities. For this to work, however, it is essential that the formal properties of the algebra that is modeled correspond appropriately to the structure of the situation that we wish to represent. Let us, then, consider the formal properties of shape algebras.

One very common approach to formalization and computer implementation of a shape algebra emerged in the earliest days of computer-aided design. It is founded on the idea that a straight line segment can be described by the coordinates of its endpoints. A shape can thus be described as a set of lines, or equivalently as a set of vertex coordinate pairs or triples together with a relation of connection in that set. Basic editing operations follow directly: lines can be added by specifying and associating endpoints, and deleted by disassociating endpoints. It also follows that translation, rotation, reflection, scaling, shearing, and perspective transformations can be performed by multiplication of coordinate vectors by transformation matrices. This idea provided a foundation for Ivan Sutherland's pioneering Sketchpad system, and a quarter of a century later it is still the basis of popular computer drafting systems such as Autocad.

The idea of describing a geometric element by specifying its boundaries can be generalized. Just as a zero-dimensional points bound one-dimensional lines, so one-dimensional lines bound two-dimensional surfaces, and two-dimensional surfaces bound three-dimensional solids. This insight provides the basis for the data structures of surface modeling systems and solid modeling systems, with their extended sets of editing operations-sweeping to create surfaces, and the spatial set operations on closed solids.

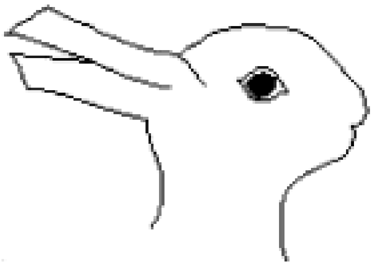
In a retrospective article, Sutherland (1975) suggested that 'the usefulness of computer drawings is precisely their structured nature." The behavior of such drawings, he noted, "is critically dependent upon the topological and geometric structure built up in the computer memory as a result of drawing operations." Traditional drawings, by contrast, have no inherent structure, and are merely "dirty marks on paper."

It is a short step from recognition that structure is important to the idea that a CAD system should automatically maintain structure as a designer manipulates a geometric model. If a designer shifts an element, for example, neighboring elements should be adjusted to maintain specified alignments and attachments. Sutherland introduced the idea of constraints that could be specified by a designer and thereafter maintained by a CAD system. Eastman (1978) later explored it in the context of three-dimensional solid modelers. More recently the idea of generalized constraint programming languages has emerged (Leler, 1988), and has found some application in computer-aided design. Gross (1989) has implemented an interesting prototype CAD system built around concepts of constraint maintenance.

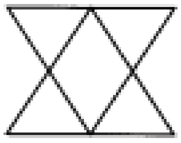
The idea of modeling in terms of geometric elements, combined with maintenance of relationships between elements, proved to be a useful and durable one. But its inherent limitations began to show up when attempts were made to use CAD systems for design exploration, rather than just representation of completed designs.

First, the structure that a designer puts into a drawing or geometric model by virtue of input operations is a limiting one. The only subshapes that it allows the designer to indicate or manipulate are subsets of the elements. Emergent subshapes are, from the computer's viewpoint, unrecognizable. Designers, however, frequently recognize emergent subshapes, and subsequently structure their understanding of the design and their reasoning about it in terms of emergent entities and relationships-ones that they never explicitly input. When this happens there is a mismatch between the way that a CAD system is explicitly structuring the design and the way that the designer is implicitly structuring it, so the explicit structure becomes a hindrance rather than a help.

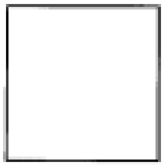
The problem is compounded by the propensity of designers to see the same shapes as different things in different contexts and on different occasions. This is most dramatic in the case of so-called ambiguous figures, such as the famous one that can be seen either as a rabbit or as a duck (but not both at once). More "creatively" it can also be seen as a front elevation of an asymmetrical Cyclops. Which structure should be maintained-that of a rabbit, that of a duck, or that of a Cyclops?



Ambiguous figures might be dismissed as bizarre anomalies but, in fact, any figure has competing readings. The following, for example, can be seen as two large triangles, or as four small triangles, or as two parallelograms, or as a pair of vertical bowties, or as a diamond bracketed by an epsilon and a reflected epsilon, or in many other ways as well (Reed 1974, Stiny 1989). There is, in fact, a substantial psychological literature on alternative structural descriptions of figures and the roles that these play in cognition (Hinton 1979, Palmer 1977).

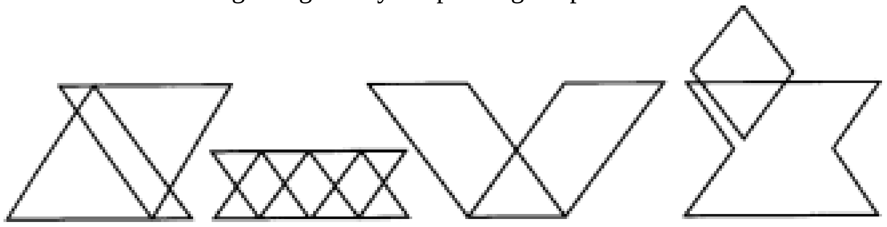


Seeing a shape as different things, I must emphasize, is not just an idle game. The act of assigning it to a class is an act of defining which of its properties are to be taken as essential and therefore maintained during design manipulations (if the shape is not unintentionally to become a thing of another kind), and which of its properties are to be taken as accidental and therefore subject to variation. If you see the following object as a square you can translate, rotate, reflect, and scale it, but you cannot change its proportions since its particular proportions are characteristic of squares. If you see it as an instance of a rectangle you can change its proportions but not its vertex angles. If you just see it as four lines you can shift and resize these lines in any way that you want.



Conversely, if you translate, rotate, reflect or scale this object you are tacitly treating it as a square. If you scale it unequally you are treating it as a rectangle. If you shear it you are treating it as a parallelogram, and if you otherwise shift and resize the lines you are just treating it as a line figure. You may want to treat it in different ways at different moments.

Different readings of shapes also induce development and refinement in different directions. What further operations are suggested by the rabbit/duck? If you see it as a duck you may (like Robert Venturi) want to substitute a decorated shed. If you see it as a rabbit you may (like Elmer Fudd) want to perform some violent transformation on the rascal. What can you make out of the 'two-triangle' figure by displacing its parts?



A further twist is given to the problem if different designers operate on the same representation-a possibility that is often claimed as an important advantage of CAD systems. What if they all see a shape as different things? What if some of them are "aspect-blind", and cannot see the shape as others see it? (Wittgenstein raised the question of aspect-blindness in Philosophical Investigations (1968), and suggested that the aspect-blind would have "an altogether different relationship to pictures from ours.") What if, by virtue of the particular structure that it imposes, the CAD system itself is significantly aspect-blind?

An impatient practical man might reply that this is all very well, but ambiguity is just a bad thing-precisely the sort of sloppiness that a CAD system, built on sound engineering principles, should root out. You should just take the commonsense course of deciding what a thing is, assigning it an appropriate structure, and sticking with that. But design is not description of what is, it is exploration of what might be. Drawings are valuable precisely because they are rich in suggestions of what might be. To blind oneself to these suggestions by imposing a rigid, Procrustean structure on a drawing is to impoverish the creative imagination.

Thus the meaning of a drawing is not adequately captured by imposing one

structure on it. The last thing that I want, as a designer, is a clanking mechanism doggedly maintaining some structure that is, for me, irrelevant and forgotten. Perhaps an analogy with word processing will make this clear. It would be absurd if your word processor allowed you to change Time flies like an arrow into Time flies like a boomerang but not (because of the syntax that you had assigned to the sentence by virtue of your input actions) into Australian *flies like a sheep*. Maintenance of structure is useless, and perhaps actively detrimental, unless a system also provides for convenient, fluid restructuring, and for parallel maintenance of alternative structures.

To my knowledge, no such systems yet exist in industrial strength, although there has been some experimentation with small-scale prototypes and it will be a formidable task to build them. But some theoretical foundations have been laid, particularly by George Stiny (1989). If we really want CAD systems to support creative design exploration (not just representation and analysis of completed designs) we will need to break away from simplistic, rigid notions of structure, and face up to the difficult problems of building systems that are sufficiently flexible and pluralistic in their handling of it.

Formalizing Design Rules

The earliest idea about computational treatment of design rules was that they could be expressed as procedures which would accept design requirements as input and produce appropriate shape information as output. Such a procedure might, for example, accept a list of rooms together with area and adjacency requirements and produce a plan that satisfied those requirements. This was consistent with the dogma of early modernism that architectural design was a matter of satisfying an established set of requirements as closely and efficiently as possible, and it was also consistent with early, procedurally-oriented approaches to programming in languages like Fortran.

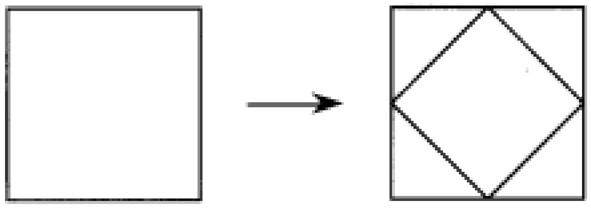
Much useful research was done within this paradigm, and sometimes this resulted in programs that actually worked. In the mid-1970s, for example, Philip Steadman, Robin Liggett and I published an efficient, rigorous procedure for producing small rectangular floor plans that satisfied adjacency, area, and dimensional constraints, and that minimized total floor area (Mitchell, Steadman, and Liggett 1976, Steadman 1983).

A fundamental limitation of this approach, however, is that it requires design rules to be expressed in a very cumbersome and artificial way-strictly in terms of the constructs provided by procedural programming languages. A clever programmer can certainly do this, but the architectural content of the resulting code is very difficult to comprehend, and this makes it difficult to subject the rule system to critical scrutiny. Furthermore, the rules are inextricably intertwined with information that specifies a strategy for applying them, so it is usually very difficult to isolate and modify them.

But human designers learn. They see the work of others, become sensitive to new issues, become aware of new possibilities, respond to criticism, and constantly modify the rules that they apply. We usually call this stylistic evolution. It may take the form of sweeping stylistic change, or that of refinement and elaboration of an established style. In any case it is an essential component of creative design, and we must provide for it in CAD systems that are seriously intended to support creative design. Further evidence for this need is provided by the experience of those who have attempted to implement rule-based design systems, and have typically found that they must spend a great deal of time tinkering with the rules to get them to produce the right sorts of results.

Adoption of a more modular and declarative style of programming is an important step in the right direction. If rules are expressed as productions, for example, they are decoupled from control information, and it becomes convenient to add rules, delete rules, and modify rules. It is possible to adapt and tune a rule system in incremental, experimental fashion. Some sort of general inference engine can be used to apply the rule system in whatever form it currently exists.

There are now many ways to set up production systems that encode design rules. You can build them in Lisp, or you can take advantage of Prolog's high-level facilities for processing rules expressed in the format of first-order logic. Numerous shells for knowledge-based systems provide facilities for expression of rules in if... then format. A shape grammar is also a production system, with the additional advantage that it expresses rules directly in terms of shapes rather than in terms of some symbolic calculus. The following rule, for example, shows one thing you can do with a square-one possible compositional move.



By sketching you can rapidly discover other things to do with squares-some of them interesting and some of them not. By remembering the interesting ones you can establish a grammar for composing squares, then you can go on to explore the language specified by that grammar. If you think of some more interesting things you can add corresponding rules to the grammar, and thus structure a new terrain for design exploration. (See Knight (1986) for a detailed discussion of the way that changing the rules changes a designer's output.)

Shape rules can be thought of as stereotyped responses to stereotyped situations, and therefore as constraints on the free play of the imagination. This was certainly the position of the romantics, who opposed the hegemony of classical rules. But the imagination needs something to play with. Though the consequence of applying a given rule to a given shape is plain to see, the consequences of recursively applying a rule system to that shape can be very surprising. The rules of a grammar are not limiting prescriptions, but tools for constructing a path from the known to the unknown-tools that can be changed if they do not seem to get you to the right place.

What sorts of shape rules should a designer use to structure paths into the unknown? Useful and interesting sets of shape rules are, I think, those that yield a lot for a little. Any set of designs can be generated in an uninteresting way by a grammar that has a rule to produce each of the designs in the set, but it is no advantage to a designer to know this sort of grammar. It is advantageous, however, to know a concise grammar, with few but powerful rules, that specifies an extensive and interesting set of designs. It has been shown that such concise yet powerful grammars can be provided for interesting and important bodies of architectural design work (Stiny and Mitchell 1978, Mitchell 1989).

Graphically-expressed shape rules are much easier for a designer to understand and criticize than, say, pages of Prolog, so there is good reason to suggest that shells for building knowledge-based design systems should take the form of shape grammar interpreters that can be programmed graphically, and that allow quick and easy modification of rules. Some interesting prototype systems of this sort have been developed (Mitchell, Liggett, and Tan 1989, Nagakura 1989, Tan 1989).

The interface details of systems for building and applying sets of design rules are less important, however, than the general point that we should focus on the expression of design rules in declarative, modular, easily-understood, and easily modifiable format. In other words, we should think of design systems as open, flexible, constantly-evolving knowledge-capture devices rather than static collections of familiar tools and dispensers of established wisdom. When we can do this I think we will see the emergence of design systems that do not just mechanically assemble banalities, but that have real style and flair.

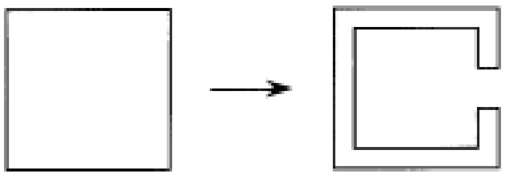
Control Strategies for Design Processes

The earliest (and still the most basic) idea here was that designing could be understood as a process of state-space search. Herbert Simon gave classic expression to this in The Sciences of the Artificial (1981)-suggesting that possession of effective heuristics for searching was an important aspect of design competence, and I emphasized it strongly in my Computer-Aided Architectural Design (1977). In his essay 'Style in Design" Simon (1975) went on to ask, "Why do we think the architect is synthesizing, or even 'creating'

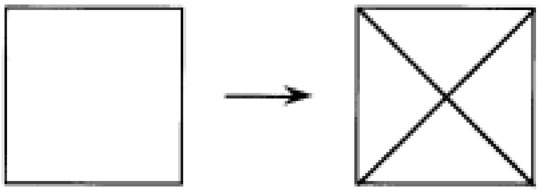
when he makes the layout?' His reply was, "Because he solves his problem by moving through a large combinatorial space in which he adds one element after another to his design ... The richness of the combinatorial space in which the problem solver moves ... is the hallmark of design creativity."
The theory of search has been investigated extensively. Basic distinctions have been made between breadth-first and depth-first search, and between top-down and bottom-up strategies. The idea has been elaborated by introduction of ideas of abstraction and planning in search spaces. Some successful architectural CAD software has been based on the idea of sophisticated searching within very large state-spaces. Some floor plan layout programs, for example, use statistical estimates of the probability of finding a good solution to choose between alternative placements (Liggett and Mitchell, 1981).

But searching is not the answer to everything. We know now that many of the most interesting design problems, when formulated for solution by search,. turn out not to be amenable to rigorous solution with a feasible amount of computation (Carey and Johnson, 1978). The attitude has grown, in the artificial intelligence community, that extensive searching is usually a bad idea-something to be attempted only when you can think of nothing better to do. Instead of doing a lot of searching guided by relatively little knowledge about a problem domain, we should emphasize limited searching guided by a lot of specialized knowledge about the particular domain of interest.

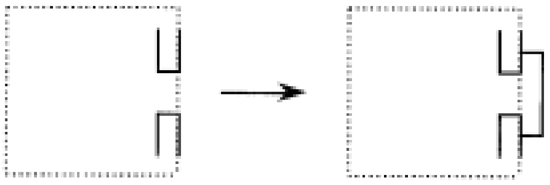
What, specifically, does this general principle (now almost a platitude) mean in the context of design? My speculation is that a designer needs to know where to direct attention in an evolving design, and needs some mechanism for indexing knowledge by shape so that it can be evoked by paying attention to a particular conditions in a design. The left-sides of shape rules in a shape grammar can be thought of as definitions of the conditions that a designer should watch out for and possibly pay attention to as a design develops, and the right-sides specify responses that can be evoked for consideration by the occurrence of particular conditions. Typically (as in the example below), left-sides will consist of patterns of abstractions and construction lines, while rightsides tell how to develop these, in various different directions, into more detailed and explicit depictions of elements and subsystems.



Alternative right-sides associated with a given left side specify the different things that an abstraction might be developed into. By adding the following rule to a grammar, for example, we record the suggestion that a square parti might be developed either into a room (by the rule above) or into a pyramid.



My experience of teaching design studios has provided me with a good deal of anecdotal evidence that designers use weak and general rules for combining fairly abstract and ambiguous shapes at early stages in design processes, then use stronger and more specific rules for combining shapes with unambiguous meaning at later stages. At an early stage, for example, it might be unclear whether a given line is to be taken as a construction line or a wall of a room, and lines might be combined according to very general rules of rhythm, proportion, and symmetry. At a later stage, shapes unambiguously stand for doors, windows, walls, columns, beams, and so on, and their combination is guided by rules that encode knowledge of how these components function and how they must be interfaced to each other. The following rule, for example, tells how to relate an entry step appropriately to a door.



At early stages visual metaphors seem to play an important role. A designer might pursue the thought that a building could be like a tree, or like an elemental primitive hut, or like a human body. These metaphors provide abstract structures that can be developed towards proposals for buildings by applying rules that substitute shapes with more precise architectural reference for construction lines that have been extracted from another context.

The lesson to be learned from this is, I suggest, that skilled architectural designers use sophisticated, domain-specific control strategies when they develop rough sketches into detailed, finished designs. These strategies can be expressed, in a direct and natural way, by means of shape rules that have patterns of abstractions and construction lines on their left-sides. We will, I think, make faster progress towards the development of truly creative computer-aided design systems if we shift our focus from ideas about search, control, and planning imported from theorem-proving and game-playing to consideration of what designers actually do when they draw.

Knowing When a Design is Finished

There is a famous paradox in Plato's Meno which seems to suggest the impossibility of truly creative thought. The protagonist asks Socrates how it is ever possible to attain new knowledge. If you know what you are looking for, he suggests, it will not really be new to you when you find it. But if you don't know what you are looking for, if you can't put forth something definite as the subject of inquiry, you will have absolutely no way of ever knowing that you have found it, and the search will never terminate. Turing provided us with a modern resolution. We can establish a process for generating information to consider (propositions, or numbers, or shapes), and we can specify a test which determines whether a given piece of information is that for which we are looking-the solution to our problem. When we have generated some information that passes the test we have solved the problem, and the process terminates. More specifically, in the context of design, we can set forth the requirements that a shape must satisfy, then attempt to instantiate a shape that does so. The design is successfully completed when we can show that we have a shape which satisfies the requirements. We may never have seen this shape before we make it explicit for consideration: it may be genuinely new and surprising to us.

In the early days of CAD this was seen as a matter of stating programmatic and technical requirements, then applying analysis procedures to a database to determine whether or not these requirements were satisfied (Mayer, 1988). A floor plan layout program, for example, might incorporate procedures to test a proposed plan for compliance with given room adjacency and area requirements. This approach was a very direct extension of the traditions of engineering analysis that had begun with Galileo's analysis of designs for a cantilever beam to determine whether they would prove strong enough.

Clearly this approach was, and continues to be, very useful. But analysis procedures certainly have not supplanted human critics. Why?

One reason is that architectural designs (as opposed to some sorts of engineering designs) often have extraordinarily complex entailments. Buildings perform subtle economic, social, and cultural roles, and these can only be understood adequately by reasoning about them in the light of extensive economic, social, and cultural knowledge. It might be possible, in principle, to build artificial critics that could apply such broad-based knowledge to produce detailed, useful evaluations of design proposals, but no such systems have emerged so far. The evaluations and appraisals that CAD systems do successfully produce are extremely narrowly focused.

A second problem may be thought of as a special case of the well-known 'frame problem" in artificial intelligence (Dennett 1984). How do you know what possible consequences of a design proposal should be given attention and explored? There are indefinitely many ways that a design does not fail, but most of these are irrelevant. Why doesn't a critic spend time carefully demonstrating, to everybody's satisfaction, that painting the walls yellow won't cause the bathtub to explode? This is a perfectly valid line of inference, but not one that is worthwhile to pursue. Clearly the critic has some way of knowing which parts of the indefinitely extensive entailment of a design are worth attention and which are not, but precisely how the critic knows this is a very difficult (and I think unsolved) problem.

It might seem that the issue could be settled by setting forth an a *priori* agenda of relevant issues for consideration. In well-defined problems, such as chess problems and theorem-proving problems, it is fairly straightforward to provide a complete specification of what is required. But in creative design, characteristically, this is impossible: issues and solution criteria are evoked contextually as the design takes shape (Reitman 1965, Akin 1986). You do not necessarily know what you want until you see what you can have. Nor do you know what to avoid until you have seen some failures. So the critical agenda may evolve and change as the design possibilities structured by shape rules are made explicit, and may not, in fact, stabilize until the point of termination has almost been reached. The most original and incisive criticism is often so because it departs from established agendas, establishes new issues for consideration, and reshapes the discourse.

Furthermore, different presumptions about what a thing is meant to be, or needs to be, or just might be, lead to different diagnoses of what it lacks. If you see that a shape might be a duck you can, from the rule All ducks have feathers, derive the criticism that the proposal lacks feathers, and suggest that feathers must be added to yield a complete and satisfactory design. But if you are more interested in its potential rabbithood you can, from the rule No rabbits have feathers derive the critical conclusion that addition of feathers would not be an appropriate way to complete the design. In general, a designer's sketch is an incomplete, ambiguous, and possibly inconsistent depiction of a possible artifact. Thus it allows contradictory observations to be made and contradictory critical inferences to be drawn from these. Design development is, in large part, a matter of identifying and resolving these inconsistencies: hence the frequent critical comment that a design in its early stages is still unresolved'. It is only in the endgame of design, when the final details are being worked out within a well-established overall framework, that there is value in the mechanisms for automatically maintaining semantic integrity of designs that have so frequently been proposed for CAD systems (Borkin 1986, Eastman 1987, Kalay 1989).

A final problem, and perhaps the most important one, is that analysis programs work within a framework of strictly monotonic reasoning. They draw critical conclusions from observations of the design together with some fairly stable, consistent body of facts and rules about the world. Adding facts and rules, or new observations about the design, should never invalidate critical conclusions that have already been drawn. But interesting critical discussion, as heard at architectural juries for example, is not really much like that. In fact, it exhibits the classic hallmarks of nonmonotonic discourse (Ginsberg, 1987). Conclusions are frequently modified or retracted when critics notice things about the design that they had not noticed before, or evoke knowledge of the world that had not previously seemed relevant, or revise their beliefs in the face of argument. From the observation that a room has no windows a critic might, for example, conclude that it would be intolerably stuffy. A second critic might challenge this conclusion by observing that the room could easily be served by airconditioning. A third critic might respond that airconditioning would be too expensive, and so on. All these critical conclusions hold in the absence of information to the contrary-but such information might be on the tip of another critic's tongue.

In sum, it is unrealistic to assume that a definitive, consistent set of design requirements can always be established ahead of time in order to provide an ironclad test for a solution. A creative design proposal may, in fact, provide a challenge to established beliefs and critical agendas. We should recognize that architectural designs are tested against a rich and complex interpretational discourse that develops in parallel with the processes of establishing design rules and making design possibilities explicit.

Conclusion

Ivan Sutherland's idea of structured design representation in computer memory, the Galilean tradition of design validation by analysis for compliance with predefined criteria, and faith in stable, universal design rules, provided the foundation on which computer-aided architectural design was initially built. But we should not remain prisoners of these ideas. Close consideration of the phenomenology of design exploration and the epistemology of criticism suggests that we must embrace the possibilities of designs that have ambiguous and unstable structural descriptions, of constructive rule systems that are provisional, fluid, and mutable as we discover what they can produce, and of critical reasoning that is not bound by assumptions of monotonicity. These issues are not, I suggest, ones that arise under anomalous conditions that can safely be ignored in mainstream, 'practical" CAD systems. On the contrary, their centrality is characteristic of creative design processes.

I do not see the emergence of these complexities as cause for pessimism. The great achievement of pioneering work in CAD has been to construct a sufficiently rigorous and comprehensive theoretical framework to allow clear identification of these issues and appreciation of their importance. The challenge now is to build a new generation of CAD systems that responds to them in sophisticated ways. The language games that architects play are subtle, and require commensurately subtle instruments.

References

Akin, Omer. 1986. Psychology of Architectural Design. London: Pion.

Borkin, Harold. 1986. Spatial and Nonspatial Consistency in Design Systems.' Environment and Planning B 13: 207-222.

Dennett, Daniel. 1984. "Cognitive Wheels: The Frame Problem in A!." In Christopher Hookway (ed.), Minds, Machines and Evolution. Cambridge: Cambridge University Press.

Eastman, Charles M. 1978. "The Representation of Design Problems and Maintenance of their Structure." In Jean-Claude Latombe (ed.), Artificial Intelligence and Pattern Recognition in Computer-Aided Design. Amsterdam: North-Holland.

Eastman, Charles M. 1987. "Fundamental Problems in the Development of Computer-Based Architectural Design Models." In Yehuda E. Kalay (ed.), Computability of Design. New York: John Wiley.

Garey, M. R., and D. S. Johnson. 1978. "Strong NP-Completeness Results: Motivation, Examples, and Applications." Journal of the Association of Computing Machinery 25, no. 3: 499-508.

Ginsberg, Mathew L. 1987. Readings in Non monotonic Reasoning. Los Altos, California: Morgan Kaufmann.

Gross, Mark. 1989. "Relational Modeling--A Basis for Computer-Assisted Design." This volume.

Hinton, G. E. 1979. "Some Demonstrations of the Effects of Structural Descriptions in Mental Imagery." Cognitive Science 3: 231-250.

Kalay, Yehuda E. 1989. Modeling Objects and Environments. New York: John Wiley.

Knight, Terry Weissman. 1986. Transformations of Languages of Designs. Ph. D. dissertation, Graduate School of Architecture and Urban Planning, University of California, Los Angeles.

Leler, William. 1988. Constraint Programming Languages. Reading, Mass.: Addison-Wesley.

Liggett, Robin S., and William J. Mitchell. 1981. "Optimal Space Planning in Practice." Computer-Aided Design 13, no. 5: 277-288.

Liggett, Robin S., and William J. Mitchell. 1981. "Interactive Graphic Floor Plan Layout Method." Computer-Aided Design 13, no. 5: 289-298.

Mayer, Thomas W. 1988. "Software Tools for the Technical Evaluation of Design Alternatives." In Thomas W. Mayer and Harry Wagter (eds.), CAAD Futures 87. Amsterdam: Elsevier.

Mitchell, William J. 1977. Computer-Aided Architectural Design. New York: Van Nostrand Reinhold.

Mitchell, William J. 1989. The Logic of Architecture. Cambridge Mass.: MIT Press.

Mitchell, William J., Robin S. Liggett, and Milton Tan. 1989. "Top-Down Knowledge-Based Design." This volume.

Mitchell, William J., Philip Steadman, and Robin S. Liggett. 1976. "Synthesis and Optimization of Small Rectangular Floor Plans." Environment and Planning B 3, no. 1: 37-70.

Nagakura, Takehiko. 1989. "Shape Recognition and Transformation--A Script-Based Approach." This volume.

Palmer, S. E. 1977. "Hierarchical Structure in Perceptual Representation." Cognitive Psychology 9: 441-474.

Reed, S. K. 1974. "Structural Descriptions and the Limitations of Visual Images." Memory and Cognition 2: 329-336.

Reitman, Walter R. 1965. "Creative Problem Solving: Notes from the Autobiography of a Fugue." In Cognition and Thought. New York: John Wiley.

Simon, Herbert A. 1981. The Sciences of the Artificial. (Second edition.) Cambridge Mass.: MIT Press.

Simon, Herbert A. 1975. "Style in Design." In Charles M. Eastman (ed.), Spatial Synthesis in Computer-Aided Building Design. New York: John Wiley.

Steadman, Philip. 1983. Architectural Morphology. London: Pion.

Stiny, George. 1989. "What Designers Do that Computers Should." This volume.

Stiny, George, and William J. Mitchell. 1978. "The Palladian Grammar." Environment and

Stiny, George, and William J. Mitchell. 1978. "Counting Palladian Plans." Environment and Planning B 5, no. 2: 189-98.

Sutherland, Ivan E. 1975. "Structure in Drawings and the Hidden-Surface Problem", in N. Negroponte (ed.), Reflections on Computer Aids to Design and Architecture. New York: Petrocelli / Charter.

Tan, Milton. 1989. "Saying What it Is by What it Is Like." This volume.

Wittgenstein, Ludwig. 1968. Philosophical Investigations. Oxford: Basil Blackwell.