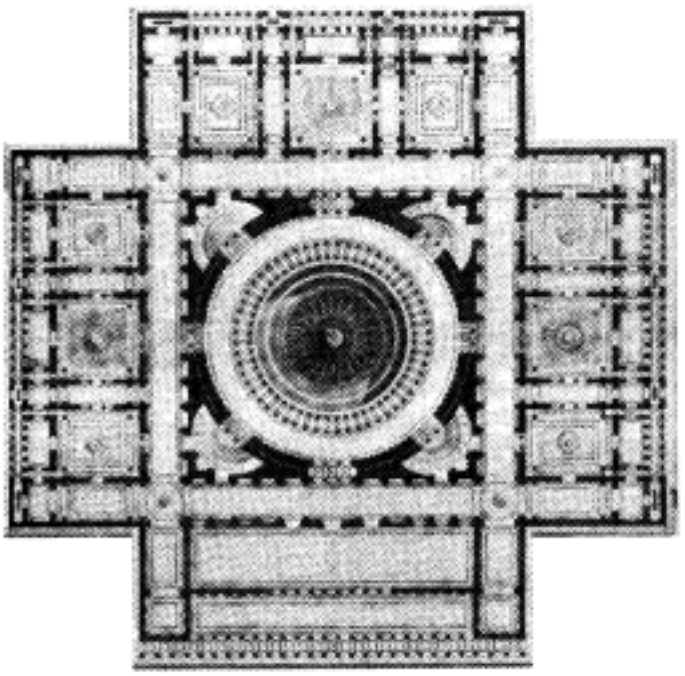
1
What Designers Do That Computers Should

George Stiny

Graduate School of Architecture and Urban Planning
University of California, Los Angeles

Designers do many things that computers don't. Some of these are bad habits that the stringencies of computation will correct. But others are basic to design, and cannot be ignored if computation is to serve creation and invention. Two of these provide the correlative themes of this paper. Both are concerned with description, and its variability and multiplicity in design.

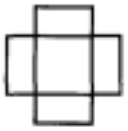
Ambiguity
In 1786 Charles Percier won the 1er Grand Prix. His design for "A Building for Assembling the Academies' was exceptional in many ways, bit especially for



its ambiguity within a fixed grid. David Van Zanten (1977) describes this in his history of *Beaux-Arts* composition.

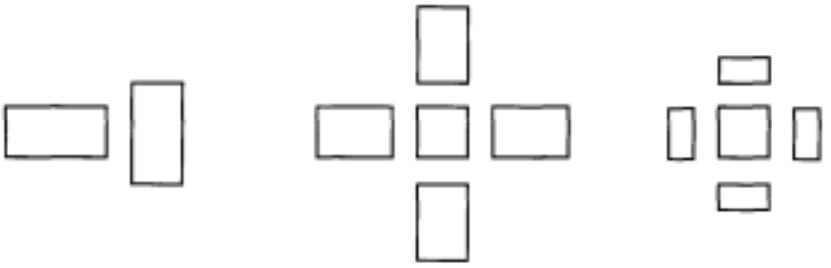
By bringing all of his spaces to rectangles, and forming them into rectangle-within-rectangle figures, Percier permitted his spaces to link together smoothly, to interpenetrate. The outer rectangle of the figure is always shared with the neighboring figure. When three such figures are set side by side, as they are laterally and longitudinally in Percier's plan, it is unclear whether one should read the resulting configuration as two interlocking rectangles, as four rectangles overlapping at the center square, or as four rectangles set around the sides of the central square. Percier thus bound his plan together and introduced a play of ambiguity through the use of the modular grid, of consistently rectangular spaces, and of the rectangle-within-rectangle figure.

From the designer's point of view, this description merely states the facts, and does so without recourse to special devices of any kind. Percier's design is a variation of the standard Greek cross

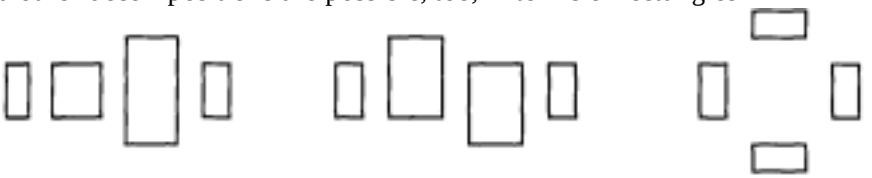


that can be decomposed into parts in many ways. If transformations are used to change the rectangles in this list

into similar ones that may be combined for this purpose, then the decompositions suggested by Van Zanten follow immediately.



And other decompositions are possible, too, in terms of rectangles



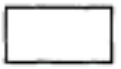
or in terms of other shapes.

Ambiguity has important uses in design where it fosters imagination and creativity, and encourages multilayered expression and response. But ambiguity is conspicuously absent from design when it is computer aided, even in the basic case where designs are given in line drawings. The reason for this is the structured nature of computer drawings, as described by Ivan Sutherland (1975).

To a large extent it has turned out that the usefulness of computer drawings is precisely their structured nature... An ordinary [designer] is unconcerned with the structure of his drawing material. Pen and ink or pencil and paper have no inherent structure. They only make dirty marks on paper. The [designer] is concerned principally with the drawings as a representation of the evolving design. The behavior of the computer-produced drawing, on the other hand, is critically dependent upon the topological and geometric structure built up in the computer memory as a result of drawing operations. The drawing itself has properties quite independent of the properties of the object it is describing.

Sutherland's enthusiasm for computer drawing is well known, and is shared widely in the CAD community. But enthusiasm alone is not enough to sustain design. It would appear that the *uselessness* of computer drawings in design is precisely their structured nature. Because the pencil and paper drawing has no inherent structure, it can be decomposed and manipulated in any manner of interest to the designer. An evolving design may thus have alternative descriptions that may change from time to time in unanticipated ways; it may be decomposed and manipulated in this way now and in another way later without difficulty. The structure of the computer drawing, however, makes all of this impossible. This structure is fixed in definite drawing operations. If the following cross

is formed by combining two rectangles congruent to this one



then it contains neither the square

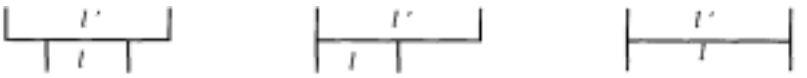
nor the rectangles

In felicitous circumstances, the two rectangles that describe the cross may coincide with a decomposition or a manipulation of interest to the designer, so that the computer may truly aid him. Otherwise, the computer will obstruct his progress. The designer cannot modify his approach or have a new idea unless his changing insights conform to the structure in the computer.

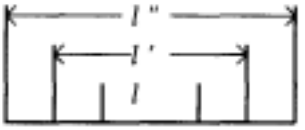
From the computer's point of view, there is no ambiguity. Every line drawing has many descriptions. Of these, the computer only recognizes the one determined as the structure of the drawing is "built up in the computer memory as a result of drawing operations". This description may or may not be of use. In the unlikely event that it is for *all* designs at *all* times, there is no reason to worry. The computer has grasped the essence of design. But if it isn't, then a change is required in CAD, from the "A" in *aided* to the "0" in *obstructed.* CAD is impossible.

This conclusion is premature. Neither the computer nor the imposed identification of a line drawing with one of its many descriptions need preclude ambiguity or obstruct design. The real problem lies elsewhere in the steadfast reliance on inappropriate topological and geometric structures in the computer to describe drawings and their parts, and to carry out drawing operations on these descriptions. Ambiguity can play its full role in CAD once the lines used by ordinary designers in drawings made with dirty marks on paper are taken seriously. Success in this effort depends on a simple relation on lines.

Every *line* is given by endpoints that are always distinct. Lines defined in this way are related according to whether one *is embedded* in another. This relation holds whenever one line is part of another, when it is a segment of it. One line can be embedded in another line in any of three basic ways



In each of these cases, the line! is embedded in the line!' In the third case, the line 1' is also embedded in the line 1, but in neither the first nor the second case is this so. If three lines 1, 1', and 1" are embedded in this way



so that I is embedded in 1' , and 1' in 1" , then 1 is also embedded in 1". As these examples show, embedding is a partial order on lines: it is transitive, and two lines are identical whenever each is embedded in the other. With lines ordered in this way, every line has a common structure given by the lines embedded in it that, with the addition of a zero assumed here only, forms a complete lattice.

Once the embedding relation is defined, shapes (drawings, and their counterparts in space) are easy to describe, so that each is identified with a definite structure in the computer that allows for arbitrary decompositions and manipulations by the designer. Every *shape is* determined by a finite but possibly empty set of maximal lines. Maximal lines are such that no two combine to form a line. When maximal lines are colinear, they are separated by a gap; otherwise, maximal lines may touch or intersect. Alone, every line is maximal. And so are the two longest lines that form these shapes

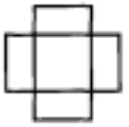
and the three longest lines that form these shapes

Of all of the different sets of lines that might be given to describe a shape, the set of maximal lines has the fewest lines in it.

This approach to shapes departs from the more customary, atomic view of things as sets of primitive components that are themselves without structure. Every shape is described uniquely by the smallest number of longest lines that combine to make it, rather than in terms of the greatest number of shortest ones-however these might be conceived-that do.

The lines given to describe a shape are maximal, but they may still have other lines embedded in them. When these lines are maximal in combination, they describe the shapes that are parts of the shape and that can be used to decompose it. The subshape relation fixes this idea formally: one shape is a *subshape* of another shape whenever every maximal line in the first shape is embedded in some maximal

line in the second shape. The cross

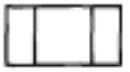


is determined by a set of eight maximal lines. It has eleven different rectangles as subshapes. Two of these are formed by combining maximal lines in the cross, and are described by subsets of the set that describes it. The other nine, though, have maximal lines that are not in the cross but that are embedded in maximal lines that are. The cross has other subshapes, too, that may be used to decompose it. Some of these are distinguished by line weight in these illustrations

The complete structure of a shape is given by all of its subshapes. These are partially ordered by the subshape relation, and form a Boolean algebra. Similar shapes have the same structure.

The subshape relation allows for shapes to be decomposed in any way whatsoever. Manipulating shapes goes forward reciprocally with two operations to combine shapes, in addition to the transformations. These operations are sum and *difference*. They correspond, respectively, to drawing and erasing lines, and are readily defined in terms of the subshape relation. A constructive approach, however, is needed for the computer; it is to be had with *reduction rules*.

The shape produced by drawing two shapes together is their sum. The shape



is the sum of a rectangle and a square

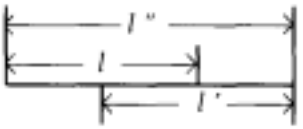
its subshapes are subshapes of the rectangle or the square, say, the three shapes

or combine subshapes of the rectangle and the square, as does the shape

Every pair of shapes A and B has a unique sum A + B given by the set of maximal lines determined according to the following three reduction rules. These rules are applied recursively in any order, first to the union of the sets of maximal lines that describe the shapes A and *B,* and then until no rule can be applied.

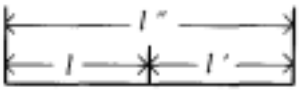
(1) If! and 1' are lines in the current set, and / is embedded in 1', then remove 1.

(2) If! and 1' are lines in the current set related to each other in this way



so that neither is embedded in the other, but a line determined by an endpoint of one and an endpoint of the other is embedded in both, then replace 1 and!' with the line 1" determined by the remaining endpoints of each.

(3) If / and 1' are lines in the current set related to each other in this way



so that neither is embedded in the other, but they share an endpoint and can both be embedded in the line!" determined by the remaining endpoints of each, then replace 1 and !' with this line.

It is interesting to notice here that the reduction rules for sum allow for shapes to be defined in schemata. This establishes families of shapes that share special features, and the basis for parametric variation in design. A shape *schema* A (x) is a set of variables. Lines are assigned to these by a function g , and may be required to satisfy certain conditions. Of course, the set g(A(x)) produced in this way may not describe a shape, as the lines in it may not be maximal. However, once the reduction rules for sum are applied to the set, the shape R(g(A(x))) is determined.

Difference is characterized in the same fashion as sum. The shape produced by erasing every subshape of one shape that is shared by another shape is the difference between these two shapes. The difference of the rectangle

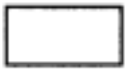
and the square

with the same centers is the shape

as the two shapes have subshapes of the shape

as their common subshapes. Conversely, the difference of the square

and the rectangle

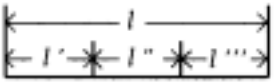


is the shape

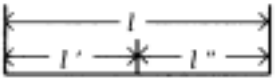
Every pair of shapes *A* and *B,* taken in this order, has a unique difference *A - B.* The set of maximal lines for this shape is determined by three reduction rules that apply recursively in any order, beginning with the set of maximal lines for the shape *A, until no rule can be applied.*

(1) If I is a line in the current set embedded in a maximal line in the shape *B,* then remove I.

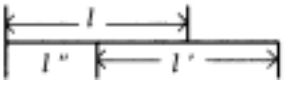
(2) Conversely, if 1 is a line in the current set related to a maximal line!' in the shape *B* in this way



so that!' is embedded in! without sharing an endpoint of!, then replace! with the lines 1" and I"', where these lines are each determined by an endpoint of! and an endpoint of!' , and do not have I' embedded in them. Alternatively, if! and I' are related in this way



so that!' is embedded in! and shares exactly one endpoint of! , then replace! with the line !" determined by the remaining endpoints of! and!'.
(3) If! is a line in the current set related to a maximal line 1' in the shape B in this way



so that neither line is embedded in the other, but a line determined by an endpoint of one and an endpoint of the other is embedded in both, then replace! with the line !", where this line is determined by an endpoint of! and an endpoint of!', and is embedded in! but not in!'.

Shapes and the operations on them form an algebra with a well known structure. The set of all shapes, and sum and difference are equivalent to a Boolean ring with a zero but no unit. If the transformations are used in this algebra, then they distribute over sum and difference. And in this case, with the trivial exception of the set containing the zero only, the algebra has no proper subset with exactly the same properties.

The algebra of shapes provides a medium for design that is equivalent in every important respect to pencil and paper. In the algebra, a shape has no inherent structure; it can be decomposed and manipulated in any way desired. But even more than this, the algebra provides for computations with shapes to be carried out by following rules, and so to define subsets of the algebra that contain shapes of certain kinds. Rules are comprised of shapes, and apply to shapes to make shapes.

With pencil and paper, one shape is changed into another shape by drawing and erasing lines. In the algebra of shapes, a sequence of these operations is defined in a rule

$$A - B$$

that replaces an occurrence of the shape A with the corresponding occurrence of the shape B. The rule applies to a shape C to produce a new shape according to the formula

$$(C - t(A)) + t(B)$$

if there is a transformation t that makes A a subshape of C. The rule exploits the full resources of the algebra in a replacement operation that is the basis of every computation with shapes. These computations are of all sorts. In fact, every computation defined in a Turing machine can be carried out equivalently by following rules in the algebra of shapes.

Like shapes, rules may be defined in schemata. This is an important generalization that combines replacement and parametric variation, and so greatly facilitates writing rules that are of broad use in design. A rule schema

$$A(x) - B(x)$$

is a pair of shape schemata $A(x)$ and $B(x)$ that may share variables. If g is a function that assigns lines to all of the variables in both of these schemata, then the rule

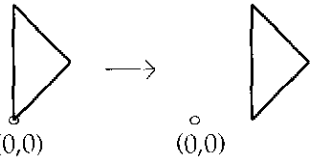
$$R(g(A(x))) - R(g(B(x)))$$

is defined.

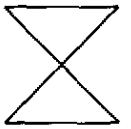
A computation is a series of shapes that is produced when rules are applied recursively. In this sense, computations with shapes are the same as computer computations with symbols. And both, too, allow for an important type of ambiguity that arises whenever there are multiple computations for the same thing,

each with equal claim to describe it. But in another sense, computations with shapes differ from symbolic computations in a very fundamental way. Every object in a symbolic computation can be resolved into its primitive components. This description is unique and is independent of the rules that may apply to the object. In contrast, no shape in a computation need correspond to any single decomposition. Its descriptions may vary discontinuously according to context, being determined by different rules and transformations. This does not mean that computations with shapes cannot he carried outby the computer, but only that special symbolic devices -the embedding relation, maximal lines, and reduction rules- are needed for this purpose.

Computations with shapes produce some surprising results that follow from discontinuities in description. The rule



simply translates an isosceles right triangle, so that its hypotenuse is moved to its right-angled corner. Intuition suggests that nothing remarkable will happen when the rule is applied. As anyone knows who has ever moved a shape from here to there, changing positions is harmless. The shape stays the same. Intuition, however, is sometimes incomplete. The rule applies first to the hourglass



and then two more times to produce the shape

But neither of the triangles in the hourglass is congruent to any subshape of this shape. Where have the triangles gone? The answer is easy to see in the shape

that is produced in the first applicaton *of* the rule. This shape has competing descriptions. In one, the triangles from the hourglass are recognized

but in the other, smaller, emergent hourglasses that are made up of subshapes of these triangles are distinguished.

If the rule is applied to translate triangles in the emergent hourglasses, then the triangles from the original hourglass disappear. More accurately, they do not exist: they are not recognized in the description of the shape

invoked when the rule is used in this way.

The usual response to this example and to a host of other ones is to dismiss them. They are frivolous and of theoretical interest only. CAD is a practical business concerned with real objects like floors, stairs, roofs, columns, walls, windows, doors, and sundry other devices that are combined in designs because they mean something. Even here, though, the present approach to computation with shapes can be recommended. Good reasons make it ineluctable.

First, discontinuities in description may arise when objects share functions and have multiple uses. This is not unknown in architecture where designs are regularly decomposed and manipulated according to unrelated categories established in different points of view, and buildings sometimes outlast their intended purposes, being adapted to new programmes. Another reason, however, addresses CAD more directly in terms of a persistent difficulty with this endeavor. There is no way to anticipate all of the devices and methods of composition that will be used in practice. What form a door takes and how it is placed in a wall in every case is the kind of question that many approaches to CAD find themselves asking. But answers to such questions are always incomplete. Practice changes, and so do ideas of doors and walls. What is needed is an approach to design that can circumscribe practice as desired and accommodate changing interests and goals within a common computational framework, so that past efforts and present enthusiasms connect. This is only possible if descriptions can vary, as in the algebra of shapes. New devices and methods of composition can be given in rules that are defined in drawings and in schemata, and that apply to all shapes in the same way, independently of how they were described originally or at any other time.

Parallel Descriptions

Ambiguity is basic to design with shapes, but does not thereby preclude descriptions of shapes or diminish their importance. Nor does it imply that all such descriptions are of equal value. In design, shapes are descriptions that may themselves be described for definite purposes, and that may serve with descriptions

of other kinds. A more complete account of design thus requires that the algebra of shapes be combined with other algebras, so that shapes and useful descriptions in different domains of interest can be computed in parallel. In this way, a *design* may be viewed as an element in an n-ary relation among shapes, other descriptions, and correlative material as needed. These relations are at the root of design, providing descriptions of things for making and accounts of how they work.

This approach is already familiar to designers. It is standard practice to combine drawings and symbols to describe form and aspects of function, material, construction, and so on. Multiple drawings are used together to establish three dimensional relationships in plans, sections, and elevations. Assemblies are broken up into smaller components that have interrelated descriptions, with details being described in ancillary documents. And analyses of one sort or another are performed on shapes and with other descriptions. In practice, drawings, symbols, documents, and analyses of many kinds connect to make a design.

Designs are complicated and multifaceted, but they are well within the realm of the computer. This is illustrated nicely with labeled shapes that allow for drawings and symbols to be combined in an integrated computational process. For this purpose, a new algebra is formed in the Cartesian product of the algebra of shapes and the algebra of sets of labeled points.

A *labeled point is* defined when a label from a specified set, say, the alphabet *a* , *b* , *c*... is associated with a point. Labels maybe just' that, in which case, they are used simply to identify and classify particular points, or they may have a semantics allowing them to carry important information. A labeled point may be transformed to produce a new one in which the same label is associated with another point. Finite sets of labeled points are partially ordered by the subset relation, and are combined by union and relative complement to form the algebra of sets of labeled points. Like the algebra of shapes, it is equivalent to a Boolean ring with a zero but no unit. And if the transformations are used in this algebra, then they distribute over both of its operations.

Sometimes, it is convenient to define sets of labeled points in schemata. Each *schema P (y)* is a set of variables. Labeled points are assigned to these by a function g, and may be required to satisfy certain conditions. Further, pairs of schemata P (y) - Q (y) define *rules* that apply to carry out computations in the algebra of sets of labeled points according to the formula given above, where A, B, and C now stand for sets of labeled points, and subshape, sum, and difference correspond to subset, union, and relative complement, respectively. And notice that these rules, too, are enough to carry out any computation defined in a Turing machine.

In a Cartesian product, the algebra of shapes and the algebra of sets of labeled points form another Boolean ring. The relation and operations of this algebra combine subshape with subset, sum with union, and difference with relative

complement. If these combinations are named by the shape relation or operation involved, then the labeled shape <A, P> is a subshape of the labeled shape <B , Q> if the shape A is a subshape of the shape *B , and* the set of labeled points P is a subset of the set of labeled points Q. And, for example, the sum of these labeled shapes has as components the sum of A and *B,* and the union of P and Q. Further, a transformation t of the labeled shape <A , P > is the labeled shape <t (A), t (P)>.

Computations in the algebra of labeled shapes are defined in *shape grammars.*

These use schemata of the form <A (x), P (y)> <*B* (x), Q (y)> to obtain rules that apply according to the formula given above, this time interpreted for labeled shapes. Values are assigned to variables in both sides of a schema by a single function, and conditions on values may relate lines and labeled points.

In a shape grammar, a computation with shapes and a computation with sets of labeled points are carried out in parallel, each influencing the other in terms of how rules in the algebra of shapes and rules in the algebra of sets of labeled points are paired in schemata in the grammar, and in terms of how conditions on values assigned to variables in these schemata relate lines and labeled points. As a result, dependencies between shapes and labeled points are established and vary as this combined computation proceeds. And conditions placed on shapes or on labeled points can be propagated between these components of the computation as the conditions are met. Labeled shapes produced in computations in the grammar define a *language* of designs that follow from a confluence of considerations, some dealing with form as described by shape, and others dealing with aspects of function, material, construction, and so on as described by sets of labeled points. The impetus for design comes from multiple perspectives that may each be dominant at different times but that are connected in computation.

It is easy to imagine different extensions of this approach in which languages of designs are defined in computations carried out by following rules in Cartesian products of appropriate algebras. For example, the algebra of labeled shapes may be combined in repeated products to define relations among plans, sections, and elevations for buildings in certain styles, or among components in assemblies of certain types. Or this algebra may be combined with an algebra of sets of labeled shapes to provide relations between labeled shapes and decompositions of them needed in different analyses. And further elaborations are also possible if labeled shapes in sets are ordered hierarchically and assigned to particular categories. Opportunities for languages of designs are vast, and can always be multiplied in yet another Cartesian product.

Designers work with descriptions involving drawings and symbols in many different ways. To do this with the computer, so that original designs can be produced in computations, requires at the very least an approach something like the one outlined in this paper. This approach is neither novel nor radical. At root, it allows for variability and multiplicity in description as they are exploited in creative

practice to be used as successfully in the computer by following rules. The approach continues a tradition in design that extends from Vitruvius to this day in which descriptive devices are invented, tried, and extended to provide an increasingly better account of past and present experience, and a more confident grasp of future possibilities. It does so without making design a science, or appealing to the computer to remake designers and what they do. Designers do just fine with pencil, paper, and imagination. The computer can do as well with shapes and symbols, their algebras, and rules.

Background

This approach to design was considered originally in Stiny (1975). More recent discussions include Stiny (1981; 1986; 1989; in preparation).

References

Stiny, G., 1975, *Pictorial and Formal Aspects of Shape and Shape Grammars,* Birkhauser, Basel.

Stiny, C., 1981, "A Note on the Description of Designs", *Environment and Planning B,* 8, 257-267.

Stiny, G., 1986, "A New Line on Drafting Systems", *Design Computing,, 1,* 5-19.

Stiny, G., 1989, "Formal Devices for Design", in S. L. Newsome, et al (eds.), *Design Theory '88,* Springer, New York.

Stiny, G., in preparation, *Shape: A Primer in Algebra, Grammar, and Description.*

Sutherland, 1., 1975, "Structure in Drawings and the Hidden-Surface Problem", in N. Negroponte (ed.), *Reflections on Computer Aids To Design and Architecture,* Petrocelli / Charter, New York.

Van Zanten, D., 1977, "Architectural Composition at the Ecole des Beaux-Arts from Charles Percier to Charles Garnier", in A. Drexler (ed.), *The Architecture of the Ecole des Beaux-Arts,* The Museum of Modern Art, New York.