Computing With Shapes and Their Boundaries

DJORDJE KRSTIC
Alcatel Internetworking, Inc

INTRODUCTION

Boundaries of shapes are repeatedly used in design to describe shapes, and are often in place of shapes themselves. We use pencil to trace a square and find no problem in considering its area as if the lines traced represent the planar shape itself. In CAD, 3D solids are often described via surface-based models. A solid is represented indirectly via its bounding surface. The surface itself is represented by the boundaries of its faces and the latter by the endpoints of their edges. This creates a hierarchy of shapes decreasing in dimension.

Boundaries are economical and simple representations because a boundary is a dimension lower than the shape it delineates. Points bound lines, lines bound planar segments and surfaces bound solids.

Using a pen to represent the boundaries of a wall in an architectural plan is more economical than using a brush to fill it in. Likewise, calculations of physical or engineering properties of shapes are often simplified when their boundaries are considered instead.

For example, many properties of 3D shapes may be expressed in terms of different volume integrals. The well-known method of direct integration may then be used to represent a volume integral over a shape via the sum of simpler surface integrals over its boundary. The closely related divergence theorem of vector calculus yields a similar result. In CAD engineering properties of solids are calculated using their bounding surfaces (Lee and Requicha, 1982; Timmer and Stern, 1980).

This work examines possibilities for representing shapes via their boundaries in the context of shape computation theory that involves shape grammars and shape algebras.

SHAPES, GRAMMARS AND ALGEBRAS

Shape grammars were introduced almost three decades ago (Stiny and Gips, 1972) as production systems capable of generating shapes to define languages of designs. Since then, shape grammars have evolved into a formal design theory that views design as a formal enterprise in which rules are adopted, and then followed to compose or describe designs.

The starting point for shape grammars is the assumption that shapes could be computed with much like it is done with numbers. The shape grammarist sees what designers do when they draw or erase shapes in the same way as an arithmetician sees adding and subtracting of numbers. That is, as computations on shapes (numbers) which produce new shapes (numbers). The mathematical notion of an algebra, as a set of objects closed under a set of operations, is broad enough to include systems that compute with numbers, like our ordinary arithmetic, as well as ones that compute with shapes. The latter systems are called shape algebras (Krstic, 1999; Stiny 1993; 1992; 1991) denoted by U_{ij}, with objects that are shapes from the set U_{ij}.

A shape from U_{ij} is a finite set of geometric elements defined in dimension i = 0, 1, 2, or 3, and manipulated in dimension j^{3} i. The elements are: points for i = 0; lines with finite, nonzero lengths for i = 1; planes with finite, nonzero areas for i = 2; or solids with finite, nonzero volumes for i = 3. Geometric elements, with the exception of points, have boundaries that are shapes in the algebra U_{i-1j}. A shape may be uniquely represented by elements that are maximal with respect to one another. These are discrete elements such that no two can be combined to produce a new element. Shapes are not constrained by their elements, again points excepted, and may be partitioned beyond their subsets. In fact, any shape that has each of its maximal elements embedded in maximal elements of another shape is a part of that shape. This partially orders shapes and leads to Boolean operations. The operations include sum and subtraction for shapes, which model the processes of drawing or erasing a shape, respectively. There is also an operation of shape product, which singles out the shape that emerges as a common part of two shapes. Such emergent shapes play an important role in the design (Stiny, 1994).

Because shapes are not only combined in design, but also moved and transformed, shape algebras include geometric transformations like rotations, translations, and reflections. The transformations form a group and may act on shapes to produce new transformed shapes. Consequently, group operations as well as a group action operation are included.

Formally, a shape algebra U_{ij} is two-sorted (Krstic 1999). It operates on objects of two different sorts: shapes from U_{ij} and transformations from set T_j. The algebra supports combinations of shapes, combinations of transformations, as well as shapes and transformations.

The hierarchy of shape algebras is given in table 1.

U_{00}	U_{01}	U_{02}	U_{03}
U_{11}	U_{12}	U_{13}	
U_{22}	U_{23}		
U_{33}			

Table 1 Hierarchy of shape algebras

For example, U_{02} manipulates shapes consisting of points from a same plane, while U_{11} manipulates collinear line segments.

A shape grammar is defined in an algebra U_{ij} by a set of rules (productions), which are pairs of shapes. A rule a ® b, where a and b are shapes, applies under a transformation t to a shape c to produce a new shape c'. The rule application takes advantage of the partial order £ relation, as well as operations group action (), difference –, and sum + of U_{ij}. First, t is distinguished so that transformation of a, t(a), becomes a part of c, t(a) £ c,
(1) then, t(a) is subtracted from c and t(b) is added to create c', c' = (c – t(a)) + t(b). (2)

A derivation of a shape in a shape grammar is a sequence of rule applications beginning with a given initial shape. If a derived shape satisfies certain conditions, it becomes an instance of a language defined by the grammar. This usually means that no rule can be applied to the derived shape.

BOUNDARIES OF SHAPES

The definition of boundaries for shapes follows directly from the fact that shape elements have boundaries.

A boundary of a shape is the set of boundaries of its elements.

Figure 1 illustrates this concept. Shape (a) defined in U_{23} may be represented by the set of planar segments (b) and its boundary by the set of their boundaries (c) defined in U_{13}. However, the same shape when represented by a different set of elements (d) yields a different boundary (e).

Boundaries as shapes may be uniquely represented with the aid of maximal elements, thus the following definition: The boundary of a shape is the set of boundaries of its maximal elements.

The set in figure 1(b) consists of maximal elements, which makes it the unique representation of the shape in figure 1(a). Consequently, the set in figure 1(c) is the unique representation of its boundary.

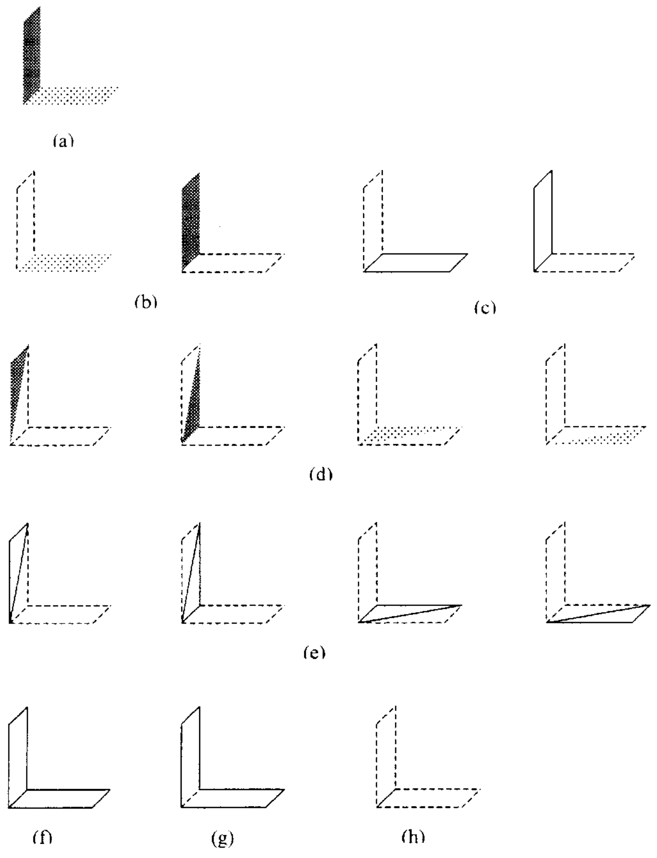

Fig. 1. Shape (a) defined in an U_{23} algebra may be represented by the set of planar segments (b) and its boundary by the set of their boundaries (c) defined in U_{13}. The same shape when represented by a different set of elements (d) yields a different boundary (e). Shapes (f), (g) and (h) are respectively the sum, symmetric difference and product of the shapes in the original boundary (c).

There is an important difference between shapes and boundaries. Although both are sets, they consist of different kinds of elements. Geometric elements that make shapes are connected entities, whereas shapes that make boundaries of elements may consist of disjoint parts. Two geometric elements combine only if they are embedded in a common element and they overlap, or their boundaries overlap. On the other hand, two shapes combine without any constraints. Geometric elements give a rise to (only) a partial algebra (Krstic, 1999), whereas shapes form an algebra.

Boundaries are sets of shapes, but they could be reduced to shapes, because any finite set of shapes is a decomposition of their sum (Krstic 1996, p. 63; Stiny, 1991). Boundaries may be defined as follows:

> The boundary of a shape is the sum of boundaries of its maximal elements.

This definition, originally due to Earl (1997) who discusses shape boundaries in detail, will be used exclusively in the remainder of this paper.

For example, the shape in figure 1(f) is the boundary of the shape in figure 1(a), thus replacing the set in figure 1(c). The shape in figure 1(f) is the sum of the shapes in the original boundary. There is, however, a loss of information due to this reduction. It is possible to reconstruct a shape from boundaries of its maximal elements, but not from the sum of the boundaries. This is illustrated in figure 2. The set of shapes (b) is the boundary of shape (a). However, shape (c), which is the sum of shapes (b) may represent the boundary of any of the four shapes (d).

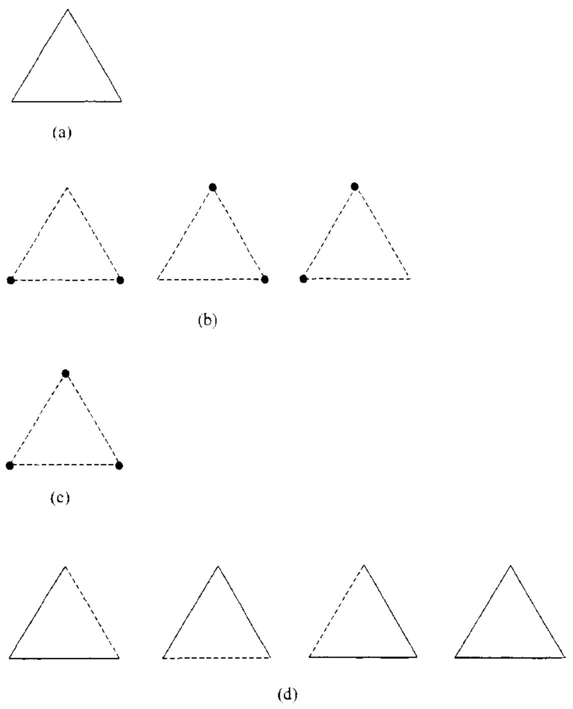

Fig. 2. Set of shapes (b) is the boundary of shape (a). Shape (c), which is the sum of shapes (b) may represent the boundary of any of the four shapes (d).

In order to handle mappings from shapes to their boundaries a new operator b: U_{ij} ® U_{i-1j} is introduced. The operator takes a shape a in U_{ij} to its boundary b(a) in U_{i-1j}. Note that the inverse of b is not a function as shown in the previous example.

Combining Shapes and Their Boundaries

Shapes paired with knowledge of some of their physical properties are usually objects of engineering design. It seems advantageous to account for boundaries whenever shape computations are carried out in an engineering design context because boundaries are instrumental in computing physical properties of shapes.

Shapes in an U_{ij} algebra have boundaries in the U_{i-1j} algebra, so that a combination of the two algebras may appear like a natural framework for simultaneous computations with shapes and their boundaries. The algebras combine forming the direct product algebra with elements that are ordered pairs of shapes from U_{ij} and U_{i-1j}, respectively. The operations of the new algebra are defined componentwise. For example, two ordered pairs are summed so that the sum of their first components is paired with the sum of their second components. The former sum is defined in U_{ij} and the latter one in U_{i-1j}. There are, however, two problems with the direct product algebra being used for computations with shapes and their boundaries.

First, not all of the ordered pairs in the algebra are of interest to us. There are many pairs with the second component that is not the boundary of the first one. Also, not all of the shapes in U_{i-1j} are boundaries of shapes in U_{ij}. There is, for example, no planar segment in U_{22} for which a line segment in U_{12} is the boundary.

Second, there is no guarantee that the boundary of, say, the sum of two shapes is the sum of their boundaries. The same is true for the difference and product. Consider, for example, two collinear lines defined in U_{11} so that they share an end point, but do not overlap. Their sum is a single line, whereas, the sum of their boundaries contains all of the three end points of the original lines. The product of the two lines is empty, but not the product of their boundaries. The latter has the common end point of the lines. The difference of the two lines is one of the lines. However, the difference of their boundaries has only one of the end points of the original difference. Note that neither the sum, nor product or difference of the boundaries is a boundary of a shape in U_{11}.

The following proposition, however, opens a door for a new kind of shape algebra that does not suffer from the above problems.

Let the binary operation of symmetric difference for shapes denoted by Å be defined by

a Å b = (a - b) + (b - a) = (a + b) - (a×b),
(3)

where a, b Î U_{ii}. Further, let an algebra U_{ii} with shape elements of dimension i defined in the space of the same dimension be denoted as a diagonal shape algebra and shape a Î U_{ii} as a diagonal shape. Then, the following proposition derived from Earl (1997) holds:

> The boundary of the symmetric difference of diagonal shapes is the symmetric difference of their boundaries.

This establishes distributivity of b over Å and can be expressed as

b(a Å b) = b(a) Å b(b),
(4)

where a, b Î U_{ii}.

Figure 3 illustrates this proposition. Shapes (a) are defined in U_{22} and have boundaries (b) defined in U_{12}. Their symmetric difference (c) has the boundary (d), which is the symmetric difference of boundaries (b).

The restriction to diagonal shapes is necessary to avoid cases where boundaries combine so that their products are parts of new boundaries. According to (3), products are never

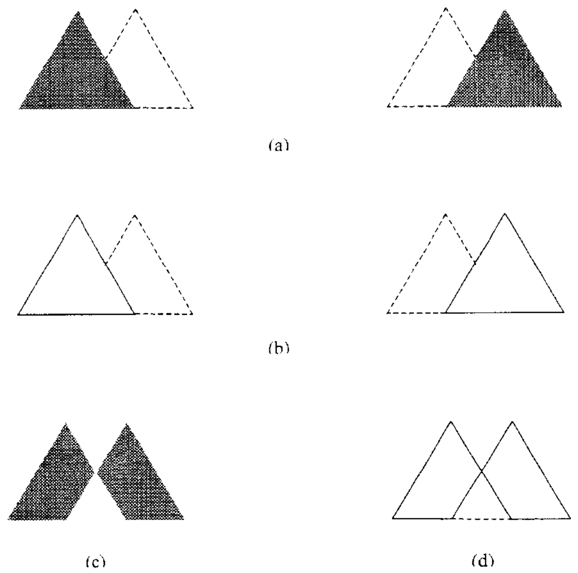

Fig. 3. Shapes (a) defined in U_{22} and have boundaries (b) defined in U_{12}. Their symmetric difference (c) has the boundary (d), which is the symmetric difference of boundaries (b).

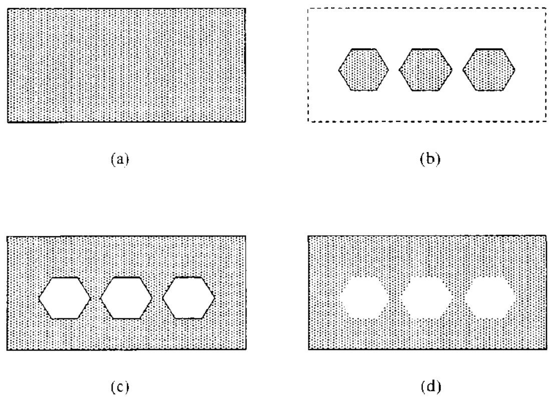

Fig. 4. Compound shapes (a) and (b) of UB_2 consisting of planar regions and their linear boundaries. The shape (c) defined in UB_2 is the symmetric difference of shapes (a) and (b), while the shape (d) defined in U_{22}'U_{12} is their difference. The former preserves the boundaries while the latter does not.

parts of symmetric differences, so (4) does not hold in such cases.

This is illustrated in figure 1. Shape (a) defined in U_{22} is the symmetric difference of shapes (b). However, the symmetric difference of their boundaries (g) does not match the boundary (f) of shape (a). It lacks shape (h), which is the product of the boundaries.

The proposition above shows that the set {(a, b(a))| a Î U_{ii}} of ordered pairs in which diagonal shapes are paired with their boundaries is closed under operation of symmetric difference. Note that symmetric difference for such pairs is defined componentwise. Consequently, this set may be elevated to a new kind of shape algebra UB_i that lacks operations of sum, product, and difference, but has a symmetric difference instead.

There are three UB_i algebras: UB_1 for collinear lines and their boundaries, UB_2 for coplanar regions and their boundaries, and UB_3 for solids that share a common 3D space and their boundaries. Points do not have boundaries so that UB_0 does not exist.

A compound shape a = (a_U, a_B) of UB_i has two components, a shape a_U and its boundary a_B = b(a_U). If i is known, then components may be referred to by their dimensions. For example, if i = 2, the first component is planar while the second one is linear.

The partial order on shapes in UB_i deserves a closer look. It is defined componentwise so that a £ b if and only if

a_U £ b_U and

a_B £ b_B,

(5)

where a = (a_U, a_B), and b = (b_U, b_B)

Shapes are usually differently related than their boundaries in which case inequalities (5) do not hold.

For example, the shape in figure 4 (b) is a part of the shape in figure 4 (a), but their boundaries are disjoint shapes. On the other hand, the boundary of the shape in figure 4 (b), is a part of the boundary of the shape in figure 4 (c), but the shapes are disjoint.

The following proposition sets conditions under which both inequalities (5) are true and a £ b:

The shape a is a part of b or a £ b if and only if a_U Í b_U.

Note that a_U and b_U are seen as sets of maximal elements rather than the shapes they represent.

It is clear that the partial order in UB_i does not rely on spatial properties of shapes. It treats shapes as sets and compares them using set inclusion Í.

GRAMMARS IN UB_I

Although UB_i is shape algebra of a kind, it is considerably weaker than U_{ij}, it combines diagonal shapes only and does it with only one operation: symmetric difference. In addition, this operation lacks the intuitive appeal of sum and difference, which model what designers do when they draw or erase shapes. Finally, the partial order in UB_i disregards spatial properties of shapes making it unintuitive from the designer's point of view. Given these severe restrictions, it is reasonable to ask if UB_i has any practical value? Can meaningful computations with shapes be carried out in UB_i and, more importantly, is it sufficient for the definition of shape grammars? Surprisingly enough, the answers to all of these questions are affirmative.

First, diagonal shapes like solids in a 3D space and planar segments in a plane are expressive enough for most if not all design (engineering) applications. The former pertain to models designers build, including 3D CAD models, while the latter relate to 2D drawings, which are traditional representations in design.

Second, depending on the context, symmetric difference may play the role of sum or difference which are both well suited for design applications. According to definition (3), symmetric difference becomes a sum if shapes are disjoint, or a difference if one shape is a part of another.
Let a and b be two shapes then

a Å b = a + b, if a×b = 0 ,
a Å b = a - b, if a ³ b and
a Å b = b - a, if a £ b .

(6)The example, in figure 4, shows compound shapes of UB_2 consisting of planar regions and their linear boundaries. The shape (a) represents a block while (b) represents a pattern of holes to be punched in it. Subtracting the latter shape from the former one should produce the desired result. However, the planar component of shape (b) is a part of the planar component of shape (a) so that in accordance with (6), symmetric difference may be used in the place of difference. Consequently, the computation may be carried out in the framework of UB_2. The result (c) preserves the boundaries of shapes. When the same computation is duplicated using difference, the result (d) has the correct planar component, but its linear component lacks the boundaries of the holes.

Another example in figure 5 shows shapes of UB_2 combined with symmetric difference acting as a sum. The shape (a) is the plan of a single space building, which is to be divided with a wall (b). This calls for a sum. However, the planar components of shapes (a) and (b) are disjoint and symmetric difference may be used instead. The result (c), as in the previous example, preserves boundaries of shapes. However, the result (d) of the same computation carried out using sum

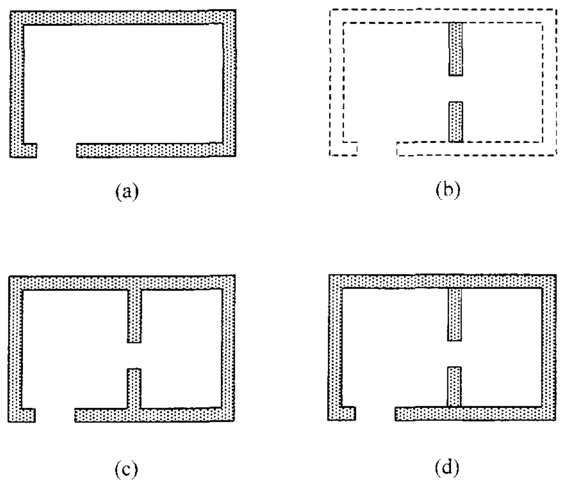

Fig. 5. Compound shapes (a) and (b) of UB_2 consisting of planar regions and their linear boundaries. The shape (c) defined in UB_2 is the symmetric difference of shapes (a) and (b), while the shape (d) defined in U_{22}'U_{12} is their sum. The former preserves the boundaries while the latter does not.

has only the planar component that is correct. The linear component of (d) has more than is needed; namely, the lines at both ends of the new wall, which prevent it from blending with the old walls.

It was shown above that subtracting and adding shapes is possible in UB_i. On the other hand, these two operations together with the part relation are sufficient for a shape grammar definition. A special kind of shape grammar defined in UB_i will be presented in the remainder of this paper.
Assume that one is interested in computing with diagonal shapes from U_{ii}, but also wants to keep track of their boundaries. The following simple program should provide for this.

-define rules using shapes from U_{ii},
-pair the shapes with their boundaries in compound shapes from UB_i,
-for each rule application determine the matching transformation in the framework of U_{ii}, but
-do the computations in UB_i.

The boundaries of shapes in computations are automatically updated.
More formally, let compound shapes a = (a_U, a_B) and b = (b_U, b_B) be defined in UB_i and let a rule be given in U_{ii} by a_U ® b_U. According to (1), the rule may apply to a compound shape c under a transformation t if the following condition

t(a_U) £ c_U,

(7) is satisfied. Then, according to (2), a new shape c' may be produced in the following two steps.

The shape t(a_U) is subtracted from c_U. According to (7) the former shape is a part of the latter one. Consequently, difference in (2) may be replaced with symmetric difference, in accordance with (6). The computation may then proceed in UB_i resulting in the shape c Å t(a).
The shape t(b_U) is then added to the shape c_U Å t(a_U). If the two are disjoint, or

t(b_U)×(c_U Å t(a_U)) = 0,

(8) sum becomes equivalent to symmetric difference allowing again for the computation to take place in UB_i. A computation that transforms c into c' is defined by

c' = (c Å t(a)) Å t(b)

(9) in UB_i. This guarantees that the boundaries of shapes are accounted for.

The new grammars with rule applications defined by (7), (8) and (9) differ from traditional shape grammars and production systems in general in two main ways.

First, the new grammars are defined in two different algebras whereas traditional grammars are given in one algebra. Rules and conditions of the new grammars are defined in U_{ii} while computations are defined in UB_i.

Second, conditions imposed on the transformation t are more restrictive than in traditional grammars. Condition (7) is equivalent to (1), but an additional condition (8) has no equivalent in traditional grammars. The latter condition involves not only the left-hand side of a rule, as is common in shape grammars and production system in general (Gips and Stiny, 1980), but also the right-hand side of the rule.

Conditions (7) and (8) are equivalent to conditions for structure rewriting rules of structure grammars (Carlson at al, 1991), which will be discussed later.

SOME NEW TYPES OF GRAMMARS

Although condition (8) may appear to be introduced for a formal reason, to allow for Å to take the place of ÷, it is not without intuitive appeal. It prevents collisions between the transformed right-hand side of a rule and the shape that remains after the transformed left-hand side of the rule is removed from the design. This may apply in situations where shapes are generated within the context of an assembly and collisions have to be avoided. Such situations are common in mechanical engineering and some recent CAD systems feature collision protection (detection). This motivates the following extension of shape grammars:

Collision protecting grammars defined in an U_{ij} algebra are shape grammars with rule applications given by the condition

c ×t(a + b) = t(a)

(10) and the formula

c' = (c - t(a)) + t(b) = (c Å t(a)) Å t(b) = c Å t(a Å b),

(11) where a, b, c, c' Î U_{ij}, t Î T_j and a ® b is a rule.

Note that condition (10) is equivalent to conditions (7) and (8) combined.

Whenever a collision protecting grammar is defined in a diagonal algebra U_{ii}, computations (11) may proceed in UB_i to account for shape boundaries.

A special kind of collision protecting grammar may be defined based on one of two basic types of shape rules. Knight (1994, p. 49), distinguishes two types of rules: addition rules, corresponding to the operation of adding a shape, and subtraction rules corresponding to the operation of subtracting a shape. The former emerge when a £ b and the latter when b £ a, where a and b are shapes and a ® b is a rule. An addition rule adds a shape t(b – a) to the design while a subtraction rule removes t(a – b) from the design.

- If all of the rules in a grammar are addition rules, the grammar is additive as defined by Knight (1999). Similarly:
- If all of the rules in a grammar are subtraction rules, the grammar is subtractive.
- If a subtraction rule a ® b satisfies condition (1) of traditional shape grammars, it automatically satisfies condition (10) of collision protecting ones. The latter condition becomes c×t(a) = t(a), which is equivalent to (1), because b £ a so that a + b = a.

Subtractive grammars are also collision protecting, and may simultaneously compute with diagonal shapes and their boundaries. Subtractive grammars may apply to model machining processes in which parts are manufactured by gradually removing material from a block. Brown et al. (1995) describes such a grammar.

Grammars that model assembly processes may have some other restrictions attached together with collision protecting. For example, relations between successive shapes in a derivation with such a grammar may be constrained. The shapes may be required to grow larger with each step of the derivation because objects they represent grow larger with each part assembled in the process. The shape to which a rule is applied becomes a part of the shape that remains after the rule application has taken place or c £ c'. The latest condition is satisfied by additive grammars (Knight, 1999), which renders additive collision protecting grammars suitable for modeling assembly processes.

Methods used to introduce collision protecting, additive and subtractive grammars may also be used to define (examine) some other types of grammars. Different grammars may arise from the different rule application conditions as well as from the different restrictions placed on the rules.

For example, general (traditional) grammars are distinguished by condition (1) under which the rules apply. Collision protecting grammars make use of another condition (10). Neither of the two types of grammars restricts the rules, while additive and subtractive grammars restrict the rules to addition and subtraction rules, respectively.

An elegant way for characterizing the above grammars, as well as some of their combinations, makes use of the condition under which rules apply. For a rule a ® b applied to a shape c under a transformation t, this condition may take a general form:

c×t(x) = t(a),

(12) where x is a combination of shapes a and b that depends on the type of the grammar.

For x = a, condition (12) becomes condition (1) for general grammars, and for x = a + b, it becomes condition (10) for collision protecting grammars.

In additive grammars a £ b, thus both a = a×b and a + b = b hold. Consequently, x = a for general grammars may become x = a×b, and x = a + b for collision protecting grammars may become x = b. That is, for general additive grammars x may be any shape between a×b and a, while for their collision protecting counterparts x is between b and a+b.

Similarly, in subtractive grammars b £ a and a + b = a, so that x between a and a + b characterizes such grammars in both the general and the collision protecting cases. Assigning combinations of a and b to x in an exhaustive fashion gives rise to yet another type of grammar.

Combinations a - b and a Å b characterize a new type of grammar featuring rules with disjoint shapes, or a×b = 0. Although it is not clear how practical or expressive these new disjoint grammars are, they are included here for the sake of completeness. General disjoint grammars have x between a - b and a whereas their collision protecting counterparts are characterized by x between a Å b and a + b.

An alternative way of characterizing grammars relies on the format of the earlier mentioned structure grammars (Carlson at al, 1991). The latter are rewriting systems capable of computing with designs and defining design languages. Structure grammars manipulate structures, which are finite sets of ordered pairs of shapes, and transformations. This is done with the aid of set operations, like in Stiny's (1982) set grammars. However, structure grammars rely on a different rule format than shape and set grammars. When adopted for shapes, the format may yield a new type of shape grammar that subsumes both general and collision protecting grammars.

Biconditional grammars defined in U_{ij} are shape grammars with rules of the form (a, y) ® b and rule applications given by the two conditions t(a) £ c and t(y)×c = 0, and formula c' = (c – t(a)) + t(b), where a, b, y Î U_{ij} and t Î T_j. The shapes a and y are, respectively, the inclusive and exclusive precedents of the rule, whereas b is the rule's consequent.

Note that a and y have to be disjoint shapes in order to satisfy the two conditions above. Consequently, the conditions become equivalent to (12) whenever x = a + y and y = x – a. The latter equation fixes y in terms of x and a, and allows for different grammars characterized by different values of x to be represented by the different biconditional grammars with corresponding values of y.

For example, because x = a characterizes general shape grammars, y = 0 characterizes their biconditional equivalents. Similarly, x = a + b characterizes collision protecting grammars whereas y = (a + b) – a = b – a characterizes their biconditional equivalents.

Table 2 provides an account of x and y values for different types of grammars.

Type of Grammar	
Unrestricted Additive a £ b S	
Subtractive b £ aDisjoint a×b = 0	
General	x = a
y = 0 a×b £ x £ a	
0 £ y £ a - b	a £ x £ a + b
0 £ y £ b - a	a - b £ x £ a
0 £ y £ a×b	
Collision Protecting	x = a + b
y = b - a	b £ x £ a + b
b - a £ y £ a Å b	a £ x £ a + b
0 £ y £ b - a	a Å b £ x £ a + b
b - a £ y £ b	

Table 2. x and y values for different types of grammars characterized by the condition c×t(x) = t(a) or, alternatively, represented by the biconditional grammars with rules of the form (a, y) ® b

Note that any general or collision protecting grammar, satisfying condition (12), may be converted into an equivalent biconditional grammar. However, only those biconditional grammars that have rules with y a combination of a and b may be converted into general or collision protecting grammars.

ACKNOWLEDGEMENTS:

I am grateful to Terry Knight and George Stiny for many useful comments and suggestions. I am particularly grateful to George Stiny for pointing out similarities between collision protecting and structure grammars.

REFERENCES:

Brown K N. McMahon C A. Sims Williams J H. "Features, a.k.a. The Semantics of a Formal Language of Manufacturing", Research in Engineering Design 7 (1995): 151-172

Carlson C. Woodbury R. McKelvey R. "An Introduction to Structure and Structure Grammars", Environment and Planning B: Planning and Design 18 (1991): 417-426

Earl C F. "Shape Boundaries", Environment and Planning B: Planning and Design 24 (1997): 669-687

Gips J. Stiny G. "Production Systems and Grammars: a Uniform Characterization", Environment and Planning B 7 (1980): 399-408

Knight T. "Shape Grammars: Six Types", Environment and Planning B: Planning and Design 26 (1999): 15-31

Knight Terry. Transformations in Design: a Formal Approach to Stylistic Change and Innovation in the Visual Arts. Cambridge University Press. 1994

Krstic D. "Constructing Algebras of Design", Environment and Planning B: Planning and Design 26 (1999): 45-57

Krstic D. Decompositions of Shapes. Ph.D. Dissertation. Los Angeles: UCLA Department of Architecture. 1996

Lee Y. Requicha A. "Algorithms for Computing the Volume and Other Integral Properties of Solids I". Communications of the ACM 25 (1982): 635-641

Stiny G. "Shape Rules: Closure, Continuity and Emergence", Environment Planning and Design 21 (1994): s49-s78

Stiny G. "Boolean Algebras for Shapes and Individuals", Environment and Planning B: Planning and Design 20 (1993):359-362

Stiny G. "Weights", Environment and Planning B: Planning and Design, 19 (1992): 413-430

Stiny G. "The Algebras of Design", Research in Engineering Design 2 (1991): 171-181

Stiny G. "Spatial Relations and Grammars", Environment and Planning B 9 (1982): 113-114

Stiny G. Gips J. "Shape Grammars and the Generative Specification of Painting and Sculpture", Information Processing_71 (1972): 1460-1465

Timmer H. Stern J. "Computation of Global Geometric Properties of Solid Objects", Computer Aided Design 10 (1980): 301-304