Environment and Planning B: Planning and Design, 1992, volume 19, pages 267–288

The maximal representation of a shape

R Krishnamurti
Department of Architecture, Carnegie Mellon University, Pittsburgh, PA 15213, USA
Received 17 December 1991

Abstract. The maximal representation of a shape is defined and algorithms for shape arithmetic are developed.

Introduction

A shape is any arrangement of spatial elements from among points, or lines, planes, volumes, or higher dimensional hyperplanes of limited but nonzero measure. According to Stiny (1991), a shape is in algebra U, or simply in U, whenever it is made up of elements from U. Thus, a shape is in U_0 if it consists of points; in U_1 if it consists of lines; and, in general, in U_n if it consists of n-dimensional hyperplanes. U_n, $n \geqslant 0$, is constructed by taking the closure under union and the Euclidean transformations[1] of any appropriate set of n-dimensional hyperplanes[2]. A shape may consist of more than one type of spatial element, in which case its algebra is given by the Cartesian product of the algebras of its spatial element types. Thus, a shape that is made up from points, lines, and planes is in $U_0 \times U_1 \times U_2$. When dealing with shapes made up of points, the points can be *distinguished* by attaching nongeometric labels; in this case, a shape consisting of labeled points is in V_0 where V_0 is the closure under union and the Euclidean transformations of a set of labels each associated with a point. Thus, a labeled shape made up of labeled points, lines, and planes is in $V_0 \times U_1 \times U_2$.

In general, a shape s is a tuple of shapes $\langle s^1, s^2, \ldots, s^k, \ldots \rangle$, where s^k is a shape in algebra U^k, $k > 0$. With the exception of sets of points, each shape s^k can be specified in indeterminately many ways as a collection of n-dimensional spatial elements, where U^k is identical to U_n. The algebras U^i and U^j, $i \neq j$, may be identical. For instance, we may choose to describe the plan, elevation, and section of a building collectively as a shape in $V_0 \times U_1 \times U_2 \times V_0 \times U_1 \times U_2 \times V_0 \times U_1 \times U_2$.

The spatial elements in shape s^k in algebra U^k, $k > 0$, may be combined with other spatial elements in the same shape s^k to form larger spatial elements. A spatial element in a shape that cannot be so combined is called a maximal spatial element. Thus, every shape s^k in U^k, $k > 0$, can be uniquely and minimally represented by its set of maximal spatial elements; if the dimensionality of U^k equals n, s^k is uniquely represented by its maximal set of n-planes. The partitioning of a shape into its distinct sets of maximal n-planes is termed the *maximal representation* of the shape.

[1] By Euclidean transformation is meant the isometric transformations augmented with scale.
[2] The set U_1 is the least set closed under union and Euclidean transformation of a line; U_2 is the least set closed under union and Euclidean transformation of all triangles, in fact, the set of all right-angled triangles will suffice. Equally, U_2 is the least set closed under union and affine transformation of any triangle. U_3 is likewise given by the least set closed under union and Euclidean transformation of the smallest set of all tetrahedra that includes all possible angle measures; in general, U_n is given by the least set closed under union and Euclidean transformation of all n-dimensional simplices defined in $n+1$ linearly independent points.

Any finite spatial element can be represented as a pair given by a *descriptor* and a *boundary*. For a spatial element, its descriptor is a measure of its orientation, and its boundary is a measure of its position and size. In principle, the descriptor of a spatial element is given by its equation; points are the exception because their descriptors are given by coordinate values. The boundary of a point is empty; the boundary of a line is given by its end-points; the boundary of a plane is given by its sets of closed connected maximal bounding lines (polygons), and so on. In general, the boundary of an n-plane, $n > 0$, is given by its sets of closed connected maximal $(n-1)$-planes where each in turn, for $n > 1$, is specified in terms of its sets of bounding connected maximal $(n-2)$-planes, and so on. That is, for any shape in U_n, $n > 0$, its boundary is a shape in U_{n-1}. Two maximal spatial elements are connected if they share a boundary element or if they are connected through a sequence of consecutively pair-wise connected elements. If the boundary of an n-plane is made up of more than one disjoint set of maximal $(n-1)$-planes, then the element contains a hole (see figure 1).

Two spatial elements of the same type can combine only when they share the same descriptor which is denoted by the functional *co*. Thus, two points can combine when they are *co*-incident; two lines can combine when they are *co*-linear; two planes can combine when they are *co*-planar; and so on. Two spatial elements that share the same *co*-descriptor are referred to as *co-equal*; otherwise, they are *co-unequal*. Two *co*-equal spatial elements combine when one element is wholly contained in the other, when one element overlaps the other; or when the two elements share boundary elements. Thus, the combination of two spatial elements is determined by relationships between their boundaries. Repeated pair-wise combinations of the spatial elements in a shape and the elements formed thereof give the maximal representation of the shape.

At this juncture, it is worthwhile to point out that the boundary of a spatial element exists only by implication. For example, the box defined by the six maximal planes shown in figure 2(a) is a shape in U_2; whereas the box defined by the twelve maximal lines shown in figure 2(b) is a shape in U_1; and the box defined by six maximal planes and twelve maximal lines shown in figure 2(c) is a shape in $U_1 \times U_2$. Thus, for the shape made up of the line l and the plane p (see figure 3), the maximal elements are l and p, even though it appears that l is part of a longer line made up of l and one of the bounding lines of p.

This distinction between defined and implied spatial elements is as real as the distinction between the physical creation of a model maker who works with sticks of balsa, as opposed to the physical creation of a model maker who works with sheets of cardboard, as opposed to the physical creation of a model maker who works with both. No other representation of spatial objects found in the literature makes coherent this distinction between a line by definition and a line by implication.

Another point to note is that, if the maximal n-planes that make up a shape in U_n, $n > 0$, are all *co*-hyperplanar and hence are all *co*-equal, the shape can be

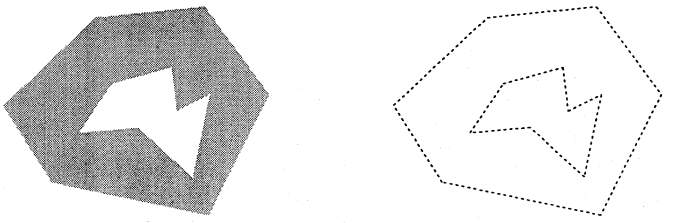


Figure 1. A maximal plane with a hole, and its boundary.

given a 'hollow' interpretation by treating it as a shape in U_{n-1}, $n > 0$, made up of maximal $(n-1)$-planes that correspond to the boundary elements of the maximal n-planes. It is for this very reason that, because every physical solid that we can see with the naked eye lies in the same three-dimensional hyperplane—namely, $w = 0$ in an (x, y, z, w)-coordinate system—we can employ the *boundary representation* of solids modeling (see Mantyla, 1988) as descriptions of real three-dimensional geometric objects. Thus, for most practical purposes, we can restrict ourselves to shapes in U_0, U_1, and U_2. Shapes in U_3 are important only if we cannot get by with a 'hollow' interpretation and require a 'solid' interpretation because arithmetic in U_3 produces different results from arithmetic in U_2. Mathematically, shapes in any U_n, $n > 0$, become more interesting when they are made up of *co*-unequal n-planes. Another feature of the maximal representation of shapes is that precisely such shapes can be dealt with in a simple, consistent, and uniform manner.

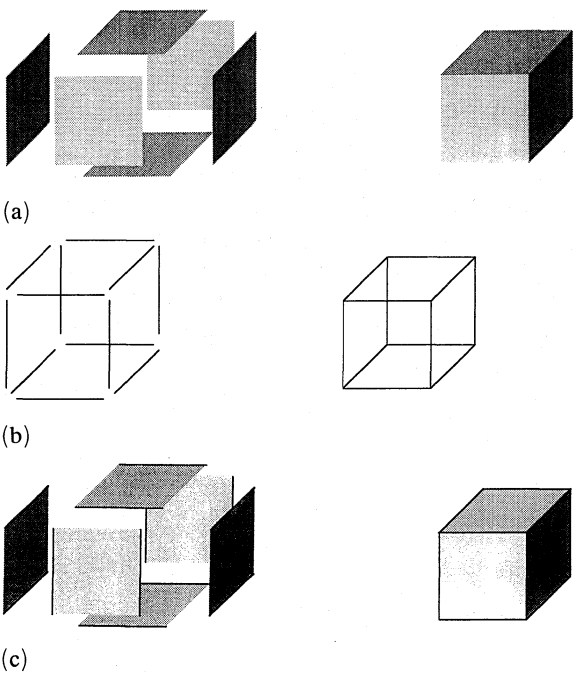


Figure 2. A box defined by (a) six maximal planes, (b) twelve maximal lines, (c) six maximal planes and twelve maximal lines.

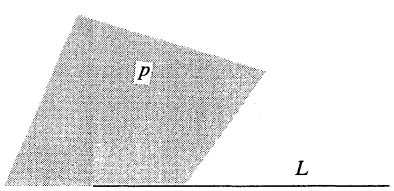


Figure 3. A maximal line colinear with a bounding line of a maximal plane.

The maximal representation of a shape

An algorithm for reducing a shape specified by its set of constituent spatial elements to its maximal representation is now given. We assume that each shape s is arranged as shapes $s^0, s^1, \ldots, s^k, \ldots; k \geqslant 0$. Further, we assume that each s^k, $k \geqslant 0$, is partititoned into equivalence classes of spatial elements according to its

co-descriptor and arranged in lexicographical order. Within each equivalence class, the spatial elements are arranged in lexicographical order according to the $(x, y, z, \ldots)$-coordinates of their maximal boundary elements. (If the specification of spatial elements by their boundary elements is not maximal, we can apply the algorithm given below to the sets of boundary elements before arranging the spatial elements in their lexicographical order.) This arrangement of the shape specification can be achieved by means of standard sorting techniques.

Thus, for any shape s_n in U_n, $n \geqslant 0$, its elements $\{e_{n,1}, e_{n,2}, \ldots, e_{n,i}, \ldots, e_{n,m}\}$, $m > 0$, satisfy either $co(e_{n,i}) \leqslant co(e_{n,j})$, $\forall i < j$, or $co(e_{n,i}) = co(e_{n,j})$ and $boundary(e_{n,i}) \leqslant boundary(e_{n,j})$, $\forall i < j$, where *boundary* is a functional which is used to arrange the elements of an equivalence class of the shape in lexicographical order according to their boundary description. One possible candidate for the boundary functional is the list of the minimum (or maximum) $(x, y, z, \ldots)$-coordinates of the *bounded sets*[3] for each element, in that order.

The algorithm is described in a series of mutually disjoint cases. Each case is expressed as a predicate with variables according to logic programming conventions[4]. A predicate is either an assertion or a rule. The control mechanism for the algorithm is based on selecting the appropriate predicate that can be satisfied through variable assignment. The scope of any variable is restricted to the predicate in which it is defined. A predicate is satisfied only when all its conditions are met and all variables defined within it have valid assignments. If no predicate can be satisfied, the algorithm fails. The algorithm halts when all variables have a valid assignment.

Algorithmic notation

A shape is expressed as an ordered arrangement of nonidentical elements within a pair of braces $\{\}$. The expression $\{e\}+s$ denotes that e is the first element in a shape and s is the shape, which may be empty, resulting from omitting e. Equivalently, $\{e\}+s$ denotes the shape consisting of the element e and the elements in shape s. In a similar fashion, $\{e, f\}+s$ denotes that e and f are the first two elements in a shape and s is the shape resulting from omitting e and f. Equivalently, $\{e, f\}+s$ denotes the shape consisting of elements e and f, and the elements in shape s. This notation extends in the obvious way. $\varnothing$ denotes the empty shape. The punctuation symbols '$\wedge$' and '$\vee$' are used as separators between conditions in a predicate, and denote logical conjunction and disjunction, respectively. ¶ indicates the start of a comment.

Maximal representation algorithm

The algorithm is executed by invoking for each s_n, $n \geqslant 0$ the recursively defined predicate *maximal* (s_n, s'_n) which is interpreted as follows: The predicate *maximal* is satisfied whenever s'_n is the maximal representation of shape s_n.

maximal$(\varnothing, \varnothing)$. (MR1)

maximal$(\{e\}, \{e\})$. (MR2)

[3] If we treat a spatial element as a point set, its *bounded set* is the least cuboid that contains the element.

[4] This avoids dealing with data-structure issues which most properly belong to discussions on algorithm efficiency and implementation. One of the lessons from computer science is that good algorithms can be made better through well-chosen data structures but not vice versa.

The two predicates state that the specification of an empty shape or of a shape consisting of just one spatial element must be maximal.

maximal($\{e, f\}+s, \{e\}+s'$) ←
  ¶ *co*-descriptor of e is less than *co*-descriptor of f
  $co(e) < co(f) \wedge$
  maximal($\{f\}+s, s'$). (MR3)

The arrow symbol ← stands for 'if'. The rule states that a shape with at least two spatial elements, e and f, has the maximal representation given by the element e and the elements in s', if e and f are *co*-unequal, and if s' is the maximal representation of the shape with element e omitted. As the elements are arranged lexicographically, $co(e) < co(f)$. It should be noted that the condition "$co(e) < co(f)$" is not a valid logic programming construct, but it does make for easier reading. The proper construct takes the form "$co(e, c_e) \wedge co(f, c_f) \wedge c_e < c_f$", which for explanatory purposes is needlessly cumbersome. I will, on occasions, resort to such shorthand notational devices.

Next, we consider the *co*-equal cases. *Co*-equality of two spatial elements implies that there is a larger element in which the two spatial elements can be embedded. The embedding relation between two spatial elements can be divided into three separate cases: (1) when one element wholly *contains* the other, (2) when the two elements *overlap* and neither contains the other; (3) when the two *co*-equal elements *share boundary* elements but nothing else. The three cases correspond to the reduction conditions stated in Stiny (1991).

maximal($\{e, f\}+s, s'$) ←
  ¶ *co*-descriptor of e equals *co*-descriptor of f
  ¶ subcase: e contains f
  $[co(e) = co(f) \wedge$ *contain*$(e, f)] \wedge$
  maximal($\{e\}+s, s'$). (MR4)

Contain is a predicate which determines whether the second element is embedded wholly within the first element. This can be determined by comparing the boundaries of the two elements. *Contain* obviously depends on the geometry of the spatial elements, and, consequently, on the algebra of the shapes under consideration. This predicate states that the shape with at least two spatial elements e and f has the maximal representation given by the elements in s', provided e and f are *co*-equal, the boundary of e contains the boundary of f, and s' is the maximal representation of the shape with f omitted. Element f can be ignored in further computation because the lexicographical ordering on the elements will forbid the situation where the boundary of f wholly contains e. In this instance, using the *boundary* functional specified above, one of the minimum coordinate values for f will have to be smaller than the minimum coordinate value for e, in which case f will occur before e in the sequence for the shape. Moreover, if f wholly contains, overlaps, or shares a boundary with some other element g in the shape s, then e must wholly contain, overlap, or share a boundary with g.

maximal($\{e, f\}+s, s'$) ←
  ¶ *co*-descriptor of e equals *co*-descriptor of f
  ¶ subcase: e overlaps with f
  $[co(e) = co(f) \wedge$ *overlap*$(e, f)] \wedge$
  ¶ e and f combine to form a larger element g
  combine$(e, f, g) \wedge$ *maximal*($\{g\}+s, s'$). (MR5)

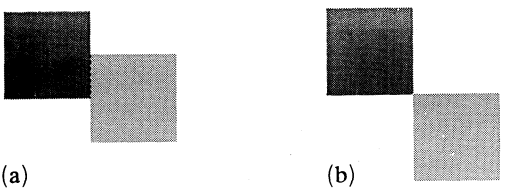
Overlap is a predicate which determines whether a nonempty part of each element is embedded in the other. As with *contain*, *overlap* compares the boundaries of the two elements, thus bringing into play the geometry of the spatial elements. *Combine* is a predicate which takes two combinable *co*-equal spatial elements and combines them into a single spatial element with the same *co*-descriptor. *Combine* and *overlap* are specific to the algebra of the shapes under consideration. This computational rule equates the original shape specification to a specification made up of the combination of the first two elements and the rest of the elements in the shape.

$$\begin{aligned} &maximal(\{e, f\}+s, s') \leftarrow \\ &\quad \P\ co\text{-descriptor of } e \text{ equals } co\text{-descriptor of } f \\ &\quad \P\ \text{subcase: } e \text{ shares boundary elements with } f \\ &\quad [co(e) = co(f) \wedge share\_boundary(e, f)] \wedge \\ &\quad \P\ e \text{ and } f \text{ combine to form a larger element } g \\ &\quad share\_combine(e, f, g) \wedge maximal(\{g\}+s, s') . \end{aligned} \tag{MR6}$$

Share_boundary is a predicate which is satisfied when the two spatial elements do not overlap but do share boundary elements. *Share_boundary* also depends on the geometry of the spatial elements and thus on the algebra of the shape under consideration. There is a connection between the definition of *share_boundary* in one algebra, and *overlap* and *contain* in the next lower dimensional algebra. *Share_combine* is a special case of *combine* that takes into account the fact that *co*-equal spatial elements share only boundary elements and not parts of the spatial elements. For two spatial elements to share a boundary there are at least two boundary elements, one from each spatial element, such that one boundary element either overlaps or contains the other. Sharing a boundary implies that the two spatial elements can be 'glued' together at boundary elements. Spatial elements cannot be so 'glued' if they only share a boundary of their boundary elements. Figure 4 illustrates this for maximal planes. Rule MR6 behaves in the same fashion as rule MR5.

The last case to consider is when the first two elements are disjoint. Disjointedness of two spatial elements can be considered in two different ways. First, the spatial elements are *co*-unequal, which is treated by rule MR3. Second, the spatial elements are *co*-equal but have no elements in common. In this case, when $n \geqslant 2$, the *boundary* functional as defined above does not preclude the situation where one element which is disjoint with the next lexicographically determined element in the shape may combine with other elements in the shape, the resultant elements then combining into larger elements. Let *disjoint* be the predicate that determines whether the boundaries of two *co*-equal elements have anything in common. *Disjoint* is specific to the algebra of the shapes under consideration. If the *boundary* functional can be defined on any set of spatial elements so that

$$\begin{aligned} &\forall i < j\ [boundary(e_{n,i}) < boundary(e_{n,j})], \\ &\quad disjoint(e_{n,i}, e_{n,j}) \Rightarrow \forall k > j,\ disjoint(e_{n,i}, e_{n,k}) , \end{aligned}$$



(a) (b)

Figure 4. Two examples of maximal planes with and without sharing boundaries: (a) two boundary lines overlap and the planes can be glued together; (b) planes share a point but not a boundary line and so cannot be glued together.

then we could add the following rule without modifying any of the other rules,

$$maximal(\{e, f\}+s, \{e\}+s') \leftarrow$$
$$[co(e) = co(f) \wedge disjoint(e, f)] \wedge$$
$$maximal(\{f\}+s, s').$$

However, the condition holds only if the set consists of lexicographically ordered maximal elements. As we are not sure whether e is maximal, we cannot ignore e in further computation. Thus, the algorithm could be improved by suitable choices for the *boundary* functional[5].

Instead, we introduce an extra predicate $reduce(e, s', s'')$ which reduces the element e with respect to the maximal representation of shape s' to produce the maximal representation of s'', the shape produced by combining e with the elements of s'. The first element of s', say g, is either f, or consists of elements that have combined with f. Notice that, because of the way the *boundary* functional is defined, the lexicographical positions of e and f (or g) will not have altered. Because e was disjoint with f, it follows that e cannot contain this new first element g of s'. Thus, only two possibilities can occur: (1) e remains disjoint with g, in which case it may still be combined with other elements in s' and any resulting combination still remains disjoint with g; (2) e may combine with g—when e overlaps with g, or when e shares a boundary with g—in which case the resulting combination may combine with other elements in s'. Because the number of elements to be considered at each level of the recursion is reduced, the procedure must eventually halt.[6] Thus, the final rule can be written as

$$maximal(\{e, f\}+s, s'') \leftarrow$$
¶ *co*-descriptor of e equals *co*-descriptor of f
¶ subcase: e is disjoint from f
$[co(e) = co(f) \wedge disjoint(e, f)] \wedge$
¶ However, e may combine with other elements in s (see figure 5)
$maximal(\{f\}+s, s') \wedge reduce(e, s', s'')$. (MR7)

Reduce is defined by the following three rules:

$$reduce(e, \{g\}+s', \{h, g\}+s'') \leftarrow$$
$$disjoint(e, g) \wedge$$
$$maximal(\{e\}+s', \{h\}+s'').$$ (MR7.1)

$$reduce(e, \{g\}+s', s'') \leftarrow$$
$$overlap(e, g) \wedge$$
$$combine(e, g, h) \wedge maximal(\{h\}+s', s'').$$ (MR7.2)

$$reduce(e, \{g\}+s', s'') \leftarrow$$
$$share\_boundary(e, g) \wedge$$
$$share\_combine(e, g, h) \wedge maximal(\{h\}+s', s'').$$ (MR7.3)

Reduce works much the same way as *maximal* (compare rules MR5–MR7), with the exception that it combines (and hence reduces) a single element with respect to a shape given by its maximal representation and which is known to be lexicographically less than all other elements in the shape. Figure 5 illustrates the possible *reduction* situations.

[5] A strict order on spatial elements for s_n, $n \geqslant 2$, would require a *boundary* functional that is not expressed simply in terms of coordinate values.

[6] It is easy to show that with the appropriate choice of data structures the algorithm can be implemented in a time linear in the number of elements in s_n.

A further improvement to the efficiency of the algorithm can be made by observing that every maximal n-plane can be transformed by an isometric transformation (for example, see Martin, 1987) to an n-plane with one of the n-coordinate values set to 0. In other words, the lexicographical ordering of the spatial elements will be either preserved or reversed, depending on whether the isometry is even (sense-preserving) or odd (sense-reversing). For instance, when $n = 1$, every line can be mapped by an isometric transformation onto the $y = 0$ line; when $n = 2$, every plane in space can be mapped onto a plane with $z = 0$ (or $x = 0$ or $y = 0$ for that matter). If the *boundary* functional returns two sets of descriptors, *min*, relating to the minimum coordinates, and *max*, relating to the maximum coordinates, then we need only compare elements whose maximum coordinates of one do not exceed the minimum coordinates of the other. In other words,

$$\forall i < j\,[max(e_{n,i}) < min(e_{n,j})],\ disjoint(e_{n,i}, e_{n,j}) \Rightarrow \forall k > j\ disjoint(e_{n,i}, e_{n,k})\,.$$

This condition holds for any set of lexicographically ordered spatial elements. Thus, once we have found the first disjoint element that satisfies the above boundary condition, we can stop comparisons with all elements in the shape with larger lexicographical indices. I leave the specification of the algorithm using the *min* and *max* boundary functionals as an exercise to the reader.

Finally, it should be noted that we do not necessarily have to fix on these seven rules. For certain values of n, because of the definitions for *overlap*, *share_boundary*, and *disjoint*, some rules may be combined into a single rule; this becomes a matter of taste and implementation. Furthermore, because *co*-unequal spatial elements in different equivalence classes can never be combined, we can effectively dispense with rule MR3 and apply the algorithm separately to each equivalence class, thus reducing the number of comparisons to be made.

This completes the enumeration of the cases and the description of the algorithm.

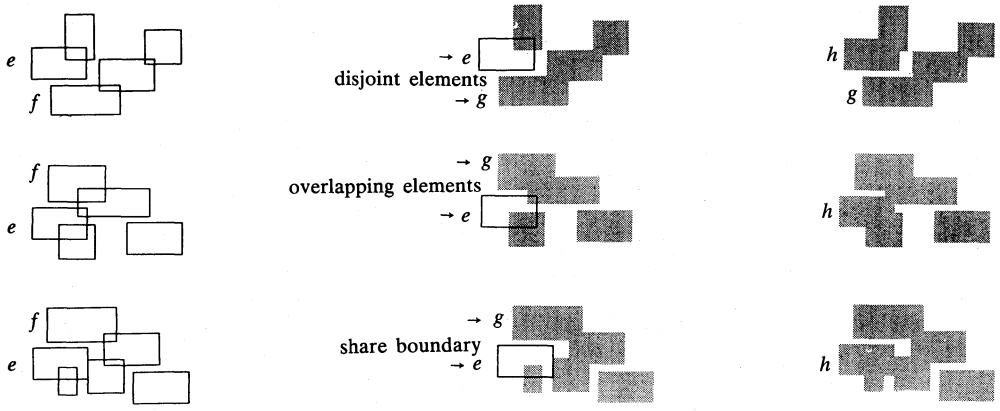


Figure 5. The possible spatial situations that can arise before and after *reduce* is invoked. The left-hand column indicates the elements of the shape whose first two elements in lexicographical order are e and f, respectively. Elements e and f are disjoint. The middle column describes the situation just before *reduce* is applied, indicating the element e and the maximal elements of the shape formed by the remaining elements whose first element is g. The right-hand column describes the situation just after *reduce* has been applied and indicates the maximal elements of the shape whose first element is h. The maximal elements of a shape are shaded the same.

Shape arithmetic

The primary purposes for developing any representation scheme are twofold. First, to use the representation as a framework for comparing objects (shapes). Second, to produce a new object (shape) from a given collection of objects (shapes). In this section, I develop algorithms for performing such operations based on the maximal representation of shapes.

Stiny (1991) has shown that for each $n \geqslant 0$, the algebra U_n satisfied the axioms of a Boolean ring under intersection and symmetric difference. Moreover, any combination of these algebras under a Cartesian product satisfies the axioms of a Boolean ring under intersection and symmetric difference. In other words, we can define the standard Boolean operations on shapes to obtain the sum of two shapes (union), the difference of two shapes (relative complement), and the shape common to two shapes (intersection). We can also define the standard Boolean relations on shapes, namely, shape equality and the subshape relation. It should be noted that the algorithms given here are not necessarily the most efficient possible, for the reason that the algorithms specified here apply to shapes defined in any algebra U_n, $n \geqslant 0$, and consequently take into account neither algebra-specific properties of shapes nor efficiencies that can be effected by selecting good data structures[7].

As before, we assume that a shape is arranged as shapes $s^1, s^2, \ldots, s^l, \ldots; l > 0$. Each s^l corresponds to a s_n, for some $n > 0$, a shape represented by a set of maximal n-planes and which is partitioned into equivalence classes based on *co*-equality.

That is,

$$s_n = \bigcup_c [e^c_{n,i}] = \bigcup_c \{e^c_{n,1}, e^c_{n,2}, \ldots, e^c_{n,m}\},$$

where $e^c_{n,i}$ is the representative element in the cth equivalence class. Because the maximal elements in an equivalence class are disjoint from each other, we have

$$\forall i < j < k,$$

$$[\mathit{boundary}(e^c_{n,i}) < \mathit{boundary}(e^c_{n,j}) \Rightarrow \mathit{disjoint}(e^c_{n,i}, e^c_{n,j})] \Rightarrow \mathit{disjoint}(e^c_{n,i}, e^c_{n,k}).$$

More algorithmic notation

We add the following notation to our algorithm description. $[e]$ denotes an equivalence class of *co*-equal elements where e is a *representative element* of the class. It should be noted that e is not any specific element but has certain properties, in particular, the shape descriptor, shared by all elements in the class. $[e]+s$ is the shape whose first element is the equivalence class with e as its representative element and s is the shape that results from omitting the elements in the equivalence $[e]$[8]. Equivalently, $[e]+s$ is the shape made up of elements in the class $[e]$ of *co*-equal elements, where e is a representative but not any specific element in the class, and the elements in shape s no element of which is *co*-equal to e. $[e]+[f]+s$ is the shape whose first two elements are the equivalence classes with e and f as their representative elements and s is the shape that results from omitting the elements in $[e]$ and $[f]$. Equivalently, $[e]+[f]+s$ is the shape made up of *co*-equal elements in classes $[e]$ and $[f]$ and the elements in shape s no element of which is *co*-equal to the elements in either class $[e]$ or class $[f]$. The notation extends in the usual way.

[7] Tuning shape algorithms to take into account algebra-specific properties of shapes is an important aspect of implementing shape arithmetic. The reader is referred to Bentley (1986), Sedgewick (1988), and Manber (1989) for insights into the art of algorithm tuning.

[8] In a programming language such as C, if the equivalence classes are represented as linked lists, list concatenation $[e]+s$ can be done with $O(1)$ pointer assignments.

The definitions for the Boolean operations on shapes and for the Boolean relations on shapes follow Stiny (1986). There, Stiny employs the *part* relation which is a partial order on shapes. A part of a shape is equivalent to any spatial element or a fragment thereof of the shape, or made up of spatial elements or fragments thereof of the shape. The empty shape is a part of every shape.

Shape union

The *union* of two shapes is the shape that just has the parts in either shape or is made up of both shapes. The union of two shapes can be computed by comparing maximal elements, two at a time, one from each shape and testing whether they combine to form a larger spatial element. The combination of two elements clearly contains the two elements and all their subelements, and thus satisfies the union definition. Moreover, we need only compare *co*-equal elements; that is, elements from equivalence classes with the same descriptor. Note that the maximal representation algorithm given previously gives us the shape union of two shapes for free if we apply the maximal representation algorithm to the ordered merged set of lines from both shapes. However, we can give an alternative algorithm that uses the ordering on the elements of the shapes implicit in their maximal representations. The algorithm is considered to be a two-stage process. First, compare the descriptors of the equivalence classes. Second, compare *co*-equal elements from classes with the same descriptor.

The first stage is described by the recursively defined predicate

$$shape\_union(s_e, s_f, s)\,,$$

which is interpreted as: the predicate *shape_union* is satisfied whenever s is the maximal representation of the union of shapes s_e and s_f.

The algorithm is enumerated in a series of cases.

First, the trivial cases, namely, if either shape is empty, their shape union is given by the other shape. That is,

$$shape\_union(\varnothing, s, s)\,. \qquad \text{(SU1)}$$

$$shape\_union(s, \varnothing, s) \leftarrow s \neq \varnothing\,. \qquad \text{(SU2)}$$

Notice that rule SU2 is expressed as a rule with the condition $s \neq \varnothing$ because the case where both shapes are empty (and hence their union is empty) is captured by rule SU1. If neither shape is empty, then each shape must have at least one equivalence class, say $[e]$ and $[f]$ respectively, which must satisfy one of three mutually exclusive conditions:

(1) $co(e) < co(f)$, (2) $co(e) > co(f)$, or (3) $co(e) = co(f)$.

For each of these cases, the rules for shape union are expressed as:

$shape\_union([e]+s_e, [f]+s_f, [e]+s) \leftarrow$
  ¶ *co*-descriptor of elements in class $[e]$ < *co*-descriptor of elements in class $[f]$
  $co(e) < co(f)$ ∧
  $shape\_union(s_e, [f]+s_f, s)$. (SU3)

$shape\_union([e]+s_e, [f]+s_f, [f]+s) \leftarrow$
  ¶ *co*-descriptor of elements in class [e] > *co*-descriptor of elements in class $[f]$
  $co(f) < co(e)$ ∧
  $shape\_union([e]+s_e, s_f, s)$. (SU4)

shape_union($[e]+s_e, [f]+s_f, [g]+s$) ←
¶ *co*-descriptor of elements in class [e] = *co*-descriptor of elements in class [f]
co(e) = *co*(f) ∧
¶ classes of elements can be combined to form a single class
co_union($[e], [f], [g]$) ∧ *shape_union*(s_e, s_f, s). (SU5)

In rule SU3, [e] can be ignored in further computation because its *co*-descriptor will be less than that of any of the remaining classes in s_e and s_f. Rule SU4 is symmetric to rule SU3. In rule SU5, we invoke the predicate *co_union* which is the second stage in the algorithm where the *co*-equal elements are compared for combination. *Co_union* returns the shape union of *co*-equal classes of maximal elements. The definition for *co_union* is

co_union($\varnothing, s, s$). (CU1)

co_union($s, \varnothing, s$) ← $s \neq \varnothing$. (CU2)

These two rules cover the trivial cases. The remainder cover the nontrivial cases.

Suppose e is the current element in the first shape and f is the current element in the second shape, then the following possibilities arise: (1) e and f are identical; (2) e wholly contains f or vice versa and they are not identical; (3) e and f overlap; (4) e and f share a boundary element; and (5) e and f are disjoint. The first two cases are captured by the following two rules:

co_union($\{e\}+s_e, \{f\}+s_f, s$) ←
¶ e contains f and e may be identical to f
contain(e, f) ∧
co_union($\{e\}+s_e, s_f, s$). (CU3)

co_union($\{e\}+s_e, \{f\}+s_f, s$) ←
¶f contains e and e is not identical to f
[*not identical*(e, f) ∧ *contain*(f, e)] ∧
co_union($s_e, \{f\}+s_f, s$). (CU4)

It should be noted that rule CU3 subsumes the case when the spatial elements are identical, because identity between spatial elements satisfies the following rule:

identity(e, f), if and only if *contain*(e, f) and *contain*(f, e).

From a practical standpoint it might be convenient to separate rule CU3 into two subcases so as to reduce the number of comparisons that need to be performed. That is, rule CU3 is equivalent to the following two rules:

co_union($\{e\}+s_e, \{f\}+s_f, \{e\}+s$) ←
identical(e, f) ∧
co_union(s_e, s_f, s). (CU3a)

co_union($\{e\}+s_e, \{f\}+s_f, s$) ←
[*not_identical*(e, f) ∧ *contain*(e, f)] ∧
co_union($\{e\}+s_e, s_f, s$). (CU3b)

Cases (3) and (4) are dealt with by the two rules:

co_union($\{e\}+s_e, \{f\}+s_f, s$) ←
¶ e overlaps with f
overlap(e, f) ∧
¶ e and f combine to form a larger element g
combine(e, f, g) ∧
co_union(s_e, s_f, s') ∧ *co_union*($\{g\}, s', s$). (CU5)

and

co_union($\{e\}+s_e, \{f\}+s_f, s$) ←
¶ e shares a boundary element with f
share_boundary(e, f) ∧
¶ e and f combine to form a larger element g
share_combine(e, f, g) ∧
co_union(s_e, s_f, s') ∧ *co_union*($\{g\}, s', s$). (CU6)

Notice that in rules CU5 and CU6, e and f are dropped from further comparison because their union g is subsequently combined with the shape union of s_e and s_f, s', to be the shape union s. This will take into account the possibility that g contains, overlaps, or shares a boundary with some elements in the shape union of s_e and s_f. The combination of two maximal planes is illustrated in figure 6.

Last, we consider the fifth case, when e and f are disjoint. Here, we first take the *co_union* of the shapes omitting e and reduce the resulting shape s' by combining its maximal elements with e. Even though e and f are disjoint, there is no reason, for $n \geqslant 2$, why e may not combine with other elements in s_f, and therefore, with elements in s'. Hence,

co_union($\{e\}+s_e, \{f\}+s_f, s$) ←
¶ e is disjoint from f
disjoint(e, f) ∧
¶ However, e may combine with other elements in s_f
co_union($s_e, \{f\}+s_f, s'$) ∧ *reduce_union*(e, s', s). (CU7)

The predicate *reduce_union* combines or reduces a single element with respect to the elements of a shape union. *Reduce_union* works much the same way as *reduce* described earlier. The reason for invoking *reduce_union* instead of directly taking the *co_union* of $\{e\}$ with s' is that the disjointedness condition of e and f does not take into account their lexicographical ordering. *Reduce_union* ensures that e is positioned correctly in the maximal representation of s. Equally, we could have reduced the shape union of $\{e\}+s_e$ and s_f by the element f to obtain the required shape union. *Reduce_union* considers the two possibilities that can arise.
(1) Element e remains disjoint with the first element, say h, of s' in which case the *boundary* functionals of e and h are compared to obtain the proper ordering on e and h. There are two points to note: first, h is disjoint with the remaining elements in s'; second, h will remain disjoint with elements in s' that combine with e.
(2) Element e combines with h and we take the shape union of the combined element with the remaining elements in s'. This situation is divided into two subcases:
(a) when e and h overlap; (b) when e and h share a boundary element.

These observations are captured in the following algorithm for *reduce_union*.

reduce_union($e, \{h\}+s_h, \{g', h'\}+s$) ←
disjoint(e, h) ∧
co_union($\{e\}, s_h, \{g\}+s$) ∧ *arrange_gh*($g, h, \{g', h'\}$). (CU7.1)

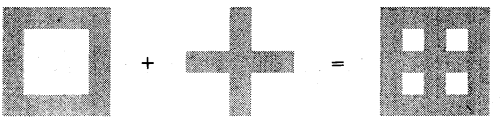


Figure 6. Two maximal planes and the resulting shape union.

and

reduce_union$(e, \{h\}+s_h, s) \leftarrow$
  overlap$(e, h) \wedge$
  combine$(e, h, g) \wedge$ *co_union*$(\{g\}, s_h, s)$. (CU7.2)

reduce_union$(e, \{h\}+s_h, s) \leftarrow$
  share_boundary$(e, h) \wedge$
  share_combine$(e, h, g) \wedge$ *co_union*$(\{g\}, s_h, s)$. (CU7.3)

Certain basic efficiency measures can be introduced into rules CU7 and CU7.1 by noting that, if the two current elements e and h are disjoint and if the maximum coordinate of e is less than the minimum coordinate of h, then e will be disjoint with the elements in s_h. Redefining the two rules by using the minimum and maximum coordinates of the spatial elements is left as an exercise to the reader.

The predicate *arrange_gh* in rule CU7.1 ensures that g and h are placed in the correct lexicographical order in the union. Note that either g equals e or g contains e. Elements g and h are disjoint with each other.

arrange_gh$(g, h, \{g, h\}) \leftarrow$
  boundary$(g) \leqslant$ *boundary*(h). (CU7.1.1)

arrange_gh$(g, h, \{h, g\}) \leftarrow$
  boundary$(g) >$ *boundary*(h). (CU7.1.2)

This completes the description of the procedure for shape union. All possible spatial conditions that can arise have been considered. Furthermore, because the application of any rule for *shape_union* and *co_union* reduces the total number of elements to be compared by at least one element, the procedure will halt. In view of the double recursion in rules CU5 through CU7 the algorithm has a worst-case time-bound proportional to the product of the number of elements in the two shapes.

Shape difference

The *difference* of two shapes is the shape that has just the parts of the first shape that, when nonempty, do not have nonempty spatial elements that are also parts of the second shape. That is, shape difference is the relativised complement of the second shape with respect to the first shape. The algorithm for shape difference follows similar lines to that for the shape union algorithm and is given as a two-stage process.

Shape difference is obtained by invoking the recursively defined predicate

shape_difference(s_e, s_f, s),

which states that: the predicate *shape_difference* is satisfied whenever s is the maximal representation of the relative complement of shape s_f with respect to shape s_e.

As before, the algorithm is enumerated in a series of cases.

shape_difference$(\varnothing, s, \varnothing)$. (SD1)

shape_difference$(s, \varnothing, s) \leftarrow s \neq \varnothing$. (SD2)

shape_difference$([e]+s_e, [f]+s_f, [e]+s) \leftarrow$
  ¶ *co*-descriptor of elements in class $[e]$ < *co*-descriptor of elements in class $[f]$
  ¶ elements of $[e]$ are in the shape difference
  $co(e) < co(f) \wedge$
  shape_difference$(s_e, [f]+s_f, s)$. (SD3)

and

shape_difference($[e]+s_e, [f]+s_f, s$) ←
  ¶ *co*-descriptor of elements in class $[e]$ > *co*-descriptor of elements of class $[f]$
  ¶ elements of $[f]$ are not in the shape difference
  $co(f) < co(e)$ ∧
  shape_difference($[e]+s_e, s_f, s$). (SD4)

shape_difference($[e]+s_e, [f]+s_f, s'$) ←
  ¶ *co*-descriptor of elements in class $[e]$ = *co*-descriptor of elements in class $[f]$
  ¶ elements of $[e]-[f]$ are in the shape difference
  $co(e) = co(f)$ ∧
  ¶ $[e]-[f]$ may be empty; hence, the *APPEND*.
  co_difference($[e]$, $[f]$, $[g]$) ∧ *shape_difference*(s_e, s_f, s) ∧
    APPEND($[g], s, s'$). (SD5)

Rules SD1 through SD4 are straightforward and follow immediately from the definition of shape difference. Rule SD5 does the real work by comparing classes of *co*-equal elements. *Co_difference* produces the shape difference $[g]$ of *co*-equal shapes $[e]$ and $[f]$. $[g]$ may be empty. The concatenation predicate *APPEND* ensures that such empty shape differences are discarded. *APPEND* has the form,[9]

APPEND($\varnothing, s, s$). (A1)

APPEND($s, \varnothing, s$) ← $s \neq \varnothing$. (A2)

APPEND($[g], s, [g]+s$) ← $s \neq \varnothing \wedge [g] \neq \varnothing$. (A3)

The predicate *co_difference* is defined as follows. *Co_difference* is divided into several cases.

First, the trivial cases when either shape is empty.

co_difference($\varnothing, s, \varnothing$). (CD3)

co_difference($s, \varnothing, s$) ← $s \neq \varnothing$. (CD4)

We next consider the possible spatial situations that arise when elements e and f in shapes s_e and s_f, respectively, are compared. The first situation is when one element contains the other. There are three cases: (1) e and f are identical; (2) e contains f but is not identical to f; (3) f contains e but is not identical to e.

When e is identical to f, then clearly e will not be in the shape difference. Because e is disjoint from every other element in shape s_e, f too will be disjoint from every other element in s_e, and therefore can be ignored in further comparisons. Hence,

co_difference($\{e\}+s_e, \{f\}+s_f, s$) ←
  ¶ e is identical to f; hence, $e-f$ is empty.
  identical(e, f) ∧
  co_difference(s_e, s_f, s). (CD5)

Suppose e is not identical to f, but e contains f. Then, the shape difference will contain elements that are part of the relative complement of f with respect to e. In general, the relative complement of a spatial element with respect to another will yield one or more relatively maximal spatial elements as shown in figure 7. Because f is wholly contained within e, it will be disjoint with every other element in s_e, and

[9] The version given here is a naive description. Efficient versions of *APPEND* are highly data-structure dependent. I adopt the convention that data-structure dependent predicates are given in upper case.

therefore can be ignored in further comparisons. However, e, and hence any relative complement $[g]$ of f with respect to e, may overlap with other elements in s_f, and therefore must be included for further comparisons. Furthermore, $[g]$ may contain elements whose *boundary* descriptor may be lexicographically greater than that of the other elements in s_e (see figure 8), and consequently, the shape difference of $[g]$ and the remaining elements of s_f have to be merged with the shape difference between the remaining elements in s_e and s_f to get the representation of the shape difference between s_e and s_f in lexicographical order. That is,

co_difference$(\{e\}+s_e, \{f\}+s_f, s) \leftarrow$
  ¶ e contains f and is not identical to it
  [*not identical*$(e, f) \wedge$ *contain*$(e, f)] \wedge$
  ¶ element ordering may be changed as a result of shape difference (see figure 8)
  ¶ hence, the *MERGE*
  complement$(e, f, [g]) \wedge$
    co_difference$([g], s_f, s') \wedge$ *co_difference*$(s_e, s_f, s'') \wedge$
    MERGE(s', s'', s). (CD6)

On the other hand, if f wholly contains e, then we have a situation similar to rule CD5, except that f is included for further comparisons:

co_difference$(\{e\}+s_e, \{f\}+s_f, s) \leftarrow$
  ¶ f contains e and is not identical to it
  [*not identical*$(e, f) \wedge$ *contain*$(f, e)] \wedge$
  co_difference$(s_e, \{f\}+s_f, s)$. (CD7)

Next, we consider the situation when e and f overlap or share a boundary. This case is similar to rule CD6, except that f may overlap or share a boundary with the other elements in s_e, and therefore has to be included for further comparisons. Notice that, when e and f share a boundary, their shape difference equals the element e.

co_difference$(\{e\}+s_e, \{f\}+s_f, s) \leftarrow$
  ¶ e and f overlap
  overlap$(e, f) \wedge$
  complement$(e, f, [g]) \wedge$
    co_difference$([g], s_f, s') \wedge$ *co_difference*$(s_e, \{f\}+s_f, s'') \wedge$
    MERGE(s', s'', s). (CD8)

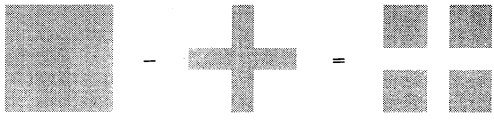


Figure 7. Two maximal planes and their resulting shape difference.

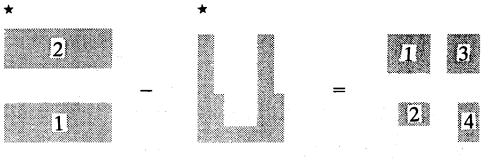


Figure 8. A possible effect on element ordering after a shape difference. ⋆ is a reference point.

and

co_difference($\{e\}+s_e$, $\{f\}+s_f$, s) ←
¶ e shares only boundary elements with f, and hence, $e-f$ is empty
share_boundary(e, f) ∧
co_difference($\{e\}$, s_f, s') ∧ *co_difference*(s_e, $\{f\}+s_f$, s'') ∧
MERGE(s', s'', s). (CD9)

The precidate *complement* in rules CD6 and CD8 takes the relative complement of one spatial element with respect to the other and its definition is algebra specific.

The last situation that has to be considered is when d and f are disjoint. We can split this into two cases: (1) when e is definitely disjoint with all elements in s_f; (2) when e may not be disjoint with the other elements in s_f. Case (1) is indicated by the condition that the maximum point of e is less than the minimum point of f, and hence by our definition of the *boundary* functional will be disjoint with all elements of s_f. Here e will be in the shape difference. Case (2) is similar to the situations expressed by rules CD6 and CD9. The two cases are captured by the following rules.

co_difference($\{e\}+s_e$, $\{f\}+s_f$, $\{e\}+s$) ←
¶ e is disjoint from f and its maximum boundary coordinate is less than that
¶ of f, and hence will be disjoint from all remaining elements in s_f
[*disjoint*(e, f) ∧ *max*(e) < *min*(f)] ∧
co_difference(s_e, $\{f\}+s_f$, s). (CD10)

co_difference($\{e\}+s_e$, $\{f\}+s_f$, s) ←
¶ e is disjoint from f and its maximum boundary coordinate is greater
¶ than that of f, and hence may not be disjoint from other elements in s_f
[*disjoint*(e, f) ∧ *max*(e) ⩾ *min*(f)] ∧
co_difference($\{e\}$, s_f, s') ∧ *co_difference*(s_e, $\{f\}+s_f$, s'') ∧
MERGE(s', s'', s). (CD11)

The routine *MERGE* employed in the description of the algorithm takes two ordered sets of spatial elements and merges them into a single ordered set. An efficient *MERGE* is data-structure dependent. The naive definition has the form:

MERGE($\varnothing$, s, s). (M1)

MERGE(s, $\varnothing$, s) ← $s \neq \varnothing$. (M2)

MERGE($[e]$, $\{f\}+s_f$, $[e]+\{f\}+s_f$) ← *max*($[e]$) < *min*(f). (M3)

MERGE($\{e\}+s_e$, $\{f\}+s_f$, $\{e\}+s$) ←
[*max*($\{e\}+s_e$) ⩾ *min*(f) ∧ *boundary*(e) < *boundary*(f)] ∧
MERGE(s_e, $\{f\}+s_f$, s). (M4)

MERGE($\{e\}+s_e$, $\{f\}+s_f$, $\{f\}+s$) ←
boundary(f) < *boundary*(e) ∧
MERGE($\{e\}+s_e$, s_f, s). (M5)

This completes the enumeration of the different spatial conditions for shape difference. The algorithm has a worst-case time-bound proportional to the product of the number of elements in the two shapes.

Shape intersection

The *intersection* of two shapes is the shape that is made up of the parts common to both shapes. The algorithm for shape intersection is similar to those for shape union and shape difference.

The shape intersection of two shapes is produced by invoking the recursively defined predicate

shape_intersection(s_e, s_f, s),

which states that: the predicate *shape_intersection* is satisfied whenever s is the maximal representation of the shape common to shapes s_e and s_f.

That is,

shape_intersection($\varnothing$, s, $\varnothing$). (SI1)

shape_intersection(s, $\varnothing$, $\varnothing$) $\leftarrow$ $s \neq \varnothing$. (SI2)

shape_intersection($[e]+s_e$, $[f]+s_f$, s) $\leftarrow$
¶ *co*-descriptor of class $[e]$ < the *co*-descriptor class $[f]$,
¶ and hence $[e]$ is not in the shape intersection
$co(e) < co(f)$ $\wedge$
shape_intersection(s_e, $[f]+s_f$, s). (SI3)

shape_intersection($[e]+s_e$, $[f]+s_f$, s) $\leftarrow$
¶ *co*-descriptor of class [e] > the *co*-descriptor class $[f]$,
¶ and hence, $[f]$ is not in the shape intersection
$co(f) < co(e)$ $\wedge$
shape_intersection($[e]+s_e$, s_f, s). (SI4)

shape_intersection($[e]+s_e$, $[f]+s_f$, s') $\leftarrow$
¶ *co*-descriptor of class $[e]$ = the *co*-descriptor class $[f]$,
¶ and hence, $[e]$ and $[f]$ may have a common shape
$co(e) = co(f)$ $\wedge$
¶ the shape intersection of $[e]$ and $[f]$ may be empty, and hence the *APPEND*
co_intersection($[e]$, $[f]$, $[g]$) $\wedge$ *shape_intersection*(s_e, s_f, s) $\wedge$
APPEND($[g]$, s, s'). (SI5)

Rules SI1 through SI4 are again straightforward and follow immediately from the definition of shape intersection. Rule SI5 compares *co*-equal elements for common elements. *Co_intersection* produces the shape, which may be empty, common to *co*-equal shapes $[e]$ and $[f]$. *APPEND* has been previously defined (see rules A1, A2, and A3).

Co_intersection is defined as follows. As before, the algorithm is divided into rules that cater for the various cases that can arise.

The trivial case when either shape is empty is handled by the following two rules.

co_intersection($\varnothing$, s, $\varnothing$). (CI1)

co_intersection(s, $\varnothing$, $\varnothing$) $\leftarrow$ $s \neq \varnothing$. (CI2)

Next we consider the case when the current elements from the respective equivalence classes contain the other. They are expressed by the following two rules. The condition "*not identical*(e, f)" in rule CI4 is necessary to ensure that only rule CI3 applies in situation when the elements are also identical.

co_intersection($\{e\}+s_e$, $\{f\}+s_f$, $\{f\}+s$) $\leftarrow$
¶ e contains f, e may be identical to f, and hence f is the common shape
contain(e, f) $\wedge$
co_intersection($\{e\}+s_e$, s_f, s). (CI3)

and

co_intersection($\{e\}+s_e$, $\{f\}+s_f$, $\{e\}+s$) ←
¶ f contains e, e is not identical to f, and hence e is the common shape
[*not_identical*(e, f) ∧ *contain*(f, e)] ∧
co_intersection(s_e, $\{f\}+s_f$, s). (CI4)

The third spatial situation occurs when the two elements e and f either overlap or share a boundary element. The rules can be stated analogously to rule CD8 and CD9. The satisfaction of rule CI5 given below relies on satisfying the algebra-specific predicate *common* which returns the common (sub)elements of the two spatial elements. When the two elements overlap, there will be at least one common subelement, denoted by the shape [g], and there may be more as illustrated in figure 9. Moreover, it is possible that the element e may overlap with other elements in the other shape. However, we need two merge operations (see rules M1 through M5) because the shapes common to e and the shape without the element f may produce an element ordering that is not consistent with the element ordering in [g]. The second merge operation is required for the same reasons as in rule CD8.

co_intersection($\{e\}+s_e$, $\{f\}+s_f$, s) ←
¶ e overlaps f
overlap(e, f) ∧
¶ element ordering may be changed as a result of shape intersection,
¶ and hence the *MERGE* operation
common(e, f, [g]) ∧
co_intersection($\{e\}$, s_f, s_e') ∧ *co_intersection*(s_e, $\{f\}+s_f$, s'') ∧
MERGE([g], s_e', s') ∧ *MERGE*(s', s'', s). (CI5)

co_intersection($\{e\}+s_e$, $\{f\}+s_f$, s) ←
¶ e only shares boundary elements with f, and hence e and f have no common shape
share_boundary(e, f) ∧
co_intersection($\{e\}$, s_f, s') ∧ *co_intersection*(s_e, $\{f\}+s_f$, s'') ∧
MERGE(s', s'', s). (CI6)

Finally, the disjoint element case. This is handled in exactly the same fashion as in shape difference (see rules CD10 and CD11).

co_intersection($\{e\}+s_e$, $\{f\}+s_f$, s) ←
¶ e is disjoint from f and its maximum boundary coordinate is less than that
¶ of f, and hence will be disjoint from all remaining elements in s_f
[*disjoint*(e, f) ∧ *max*(e) < *min*(f)] ∧
co_intersection(s_e, $\{f\}+s_f$, s). (CI7)

co_intersection($\{e\}+s_e$, $\{f\}+s_f$, s) ←
¶ e is disjoint from f and its maximum boundary coordinate is greater than
¶ that of f, and hence may not be disjoint from other elements in s_f
[*disjoint*(e, f) ∧ *max*(e) ⩾ *min*(f)] ∧
co_intersection($\{e\}$, s_f, s') ∧ *co_intersection*(s_e, $\{f\}+s_f$, s'') ∧
MERGE(s', s'', s). (CI8)

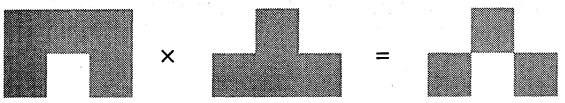


Figure 9. Shape intersection of two maximal planes to produce three maximal planes.

This completes the enumeration of all possible spatial situations and the description of the shape intersection algorithm. The worst-case performance of the algorithm is similar to that for shape union and shape difference, namely, proportional to the product of the number of elements in both shapes.

Shape equality

Two shapes are equal if their parts are identical. Alternatively, two shapes are equal if they are subshapes of each other. That is,

$$shape\_equality(s_e, s_f) \leftarrow subshape(s_e, s_f) \wedge subshape(s_f, s_e). \quad \text{(EQ1)}$$

Shape_equality succeeds only when the two shapes are equal. The predicate *subshape* is defined in the next section.

Subshape relation

A shape is a subshape of another shape if every part of the first shape is a part of the second shape. We define a predicate *subshape* (s_e, s_f) which is satisfied only when s_e is a subshape s_f. Unlike the shape operations, we need only consider the spatial situations when the subshape relation is likely to succeed.

$$subshape(\varnothing, s). \quad \text{(SS1)}$$

$$subshape([e]+s_e, [f]+s_f) \leftarrow co(f) < co(e) \wedge subshape([e]+s_e, s_f). \quad \text{(SS2)}$$

$$subshape([e]+s_e, [f]+s_f) \leftarrow co(e) = co(f) \wedge subelement([e], [f]) \wedge subshape(s_e, s_f). \quad \text{(SS3)}$$

There are only three situations to consider for the subshape relation. If the shape is empty the relationship always holds, because the empty shape is a subshape of all shapes (rule SS1). If the current class of *co*-equal elements $[e]$ has a higher descriptor value than $[f]$, then there remains the possibility that s_f contains a shape of which $[e]$ is a subshape. Hence, the truth or falsity of the subshape relationship depends on whether or not $[e]+s_e$ is a subshape of s_f (rule SS2). If the current classes are *co*-equal, then every element in $[e]$ must be a subelement of $[f]$ for the relationship to hold (rule SS3). In all other cases, the subshape relationship fails and this is indicated by the fact that no other rules are specified. In other words, the failure to satisfy a rule implies the negation of the predicate.

Subelement is similarly defined.

$$subelement(\varnothing, s). \quad \text{(SE1)}$$

$$subelement(\{e\}+s_e, \{f\}+s_f) \leftarrow identical(e, f) \wedge subelement(s_e, s_f). \quad \text{(SE2)}$$

$$subelement(\{e\}+s_e, \{f\}+s_f) \leftarrow contain(f, e) \wedge subelement(s_e, \{f\}+s_f). \quad \text{(SE3)}$$

$$subelement(\{e\}+s_e, \{f, g\}+s_f) \leftarrow [disjoint(e, f) \wedge max(e) > min(g)] \wedge subelement(\{e\}, \{g\}+s_f) \wedge subelement(s_e, \{f, g\}+s_f). \quad \text{(SE4)}$$

Rules SE1 through SE3 are obvious. Rule SE4 needs a little explanation. When element e is disjoint from f, then unless there is another element such that e is a

subelement of it, the subshape relation cannot possibly hold. Moreover, if the element g next to f, in lexicographical order, has its minimum boundary point greater than the maximum boundary point of e, then e will be disjoint with g and with all other elements in the second shape, and again the subshape relation will fail. Here too, I have omitted the cases where the subshape relation will definitely not hold. That is, negation is implied by failure to satisfy any of the rules.

Algebra-specific predicates of shapes

To complete the description of the shape algorithms for shapes in any algebra, we need to define, for each algebra, the seven algebra-specific predicates that they rely on, namely, *contain*, *overlap*, *share_boundary*, *combine*, *share_combine*, *complement*, and *common*. In the remainder of this paper I develop definitions for these predicates for the shapes in U_0 and U_1. The definitions of the basic relations and operations on shapes in U_2 are given in Krishnamurti (1991).

Shapes in U_0 and V_0

Shapes in U_0 consist of points, and in the case of V_0, points are associated with labels. The algebra-specific predicates such as *overlap*, *share_boundary*, and *contain* are equivalent to the identity relation. That is

$$overlap(e, f) \leftarrow e = f.$$

The other predicates such as *combine*, *share_combine*, *complement*, and *comment* have trivial definitions, namely,

$$combine(e, e, e).$$

$$share\_combine(e, e, e).$$

$$common(e, e, \{e\}).$$

$$complement(e, e, \varnothing).$$

By replacing the algebra-specific predicates in the algorithms for the shape operations by the above definitions and by eliminating duplicate rules, it is easy to show that shape operations on points reduce to their analogous set operations.

Shapes in U_1

The shape arithmetic for shapes in U_1 consisting of lines has been fully treated in Krishnamurti (1980) and Chase (1989). For the sake of completeness, I briefly define the algebra-specific predicates. Let a line l be represented by its endpoints $\{t(l), h(l)\}$, respectively the *tail* and *head* of line l where $t(l) < h(l)$. Then,

$$share\_boundary(e, f) \leftarrow t(e) = h(f) \vee t(f) = h(e).$$

$$contain(e,f) \leftarrow t(e) \leqslant t(h) \wedge h(f) \leqslant h(e).$$

$$overlap(e, f) \leftarrow t(e) < h(f) \wedge t(f) < h(e).$$

The *overlap* predicate as defined subsumes the containment case, but this does not cause any problems because the ordering on maximal lines is strict. That is, if e and f are two maximal lines from different shapes s_e and s_f, respectively, that combine, their combination is always disjoint from at least one of the next elements in order from s_e or s_f[10]. In fact, we can define a single predicate *combinable* to replace the three separate predicates above:

$$combinable(e, f) \leftarrow t(e) \leqslant h(f) \wedge t(f) \leqslant h(e).$$

[10] This property has been used to specify linear time shape algorithms in U_1 (see Krishnamurti, 1980).

Suppose e and f combine to form a longer line g. Then,

$$combine(e, f, g) \leftarrow min(t(e), t(f), t(g)) \wedge max(h(e), h(f), h(g)).$$

where *min* and *max* return the minimum and maximum coordinates of two points respectively. *Share_combine* has an identical definition.

If e and f overlap, their common line is given by

$$common(e, f, \{g\}) \leftarrow max(t(e), t(f), t(g)) \wedge min(h(e), h(f), h(g)).$$

and their shape difference is the set of zero, one, or two lines:

$$complement(e, f, [g]) \leftarrow line(t(e), t(f), [t]) \wedge line(h(e), h(f), [h]) \wedge APPEND([t], [h], [g]),$$

where *line* checks whether the given points observe the ordering on endpoints of a line in which case it returns a shape containing the line given by the points; otherwise, it returns an empty shape. That is,

$$line(p_t, p_h, \{\{p_t, p_h\}\}) \leftarrow p_t < p_h. \qquad line(p_t, p_h, \varnothing) \leftarrow p_t \geqslant p_h.$$

Conclusion

The maximal representation of shapes has been presented and shape arithmetic on the algebras of points, lines, and planes, the details of which are given in Krishnamurti (1991), has been developed. Several features of the maximal representation of shapes stand out.

First, the shape algorithms are general and consistent across the different algebras.

Second, there is a natural hierarchy on the algebras, wherein operations on planes can be expressed in terms of lines, which in turn can be expressed in terms of points (see Krishnamurti, 1991). Although I have not yet explicitly demonstrated this for algebras higher than that of planes, I conjecture that this hierarchy holds for any $n > 0$, namely, that operations on shapes in U_n can be expressed in terms of shape operations in U_{n-1}. This hierarchy of shapes lends itself naturally to parallel computation on shapes.

Third, the maximal representation distinguishes between defined geometric elements and implied geometric elements. The representation makes clear the distinction between maximal elements and their boundary elements. A boundary is not a basic geometric property of a shape but that of one of its geometric elements. Alter the element and one alters the shape. There is no need to describe additional shape rules for altering the boundary. These rules are incorporated in the definition of an element. Thus, the maximal representation dismisses the classical question of when a 'wire frame' model or a 'plane frame' model or a 'solid frame' model is or is not a true representation of a physically realizable object. There is no inherent notion of a dangling edge. There is no inherent notion of an infinite point set that has to be abstracted as a finite description. The representation permits individuals to define their own sense of real objects independent of any externally imposed criteria of realizability or reality.

Fourth, the geometry of spatial objects is captured in a unified manner by just four algebra-specific relations, namely, *disjoint*, *overlap*, *share_boundary*, and *contain*; and by three algebra-specific operations, namely, *combine*, *complement*, and *common*.

The representation is essentially visual in that what one defines is what one sees and vice versa. Further, the maximal representation provides a clear and clean definition of a shape as a *definite geometrical object with indefinitely many geometrical parts*, a prerequisite for any decent generative formalism for shapes.

Last, and more importantly, the maximal representation is so simple to use.

Acknowledgement. I would like to thank George Stiny for his encouragement, critical comments, and support during the course of this work. I would like to add my appreciation to Chris Earl and Rudi Stouffs for their careful reading of earlier versions of this paper and for their invaluable suggestions to improve its content.

References

Bentley J, 1986 *Programming Pearls* (Addison-Wesley, Reading, MA)
Chase S C, 1989, "Shape and shape grammars: from mathematical models to computer implementation" *Environment and Planning B: Planning and Design* **16** 215-242
Krishnamurti R, 1980, "The arithmetic of shapes" *Environment and Planning B: Planning and Design* **7** 463-484
Krishnamurti R,1991, "The arithmetic of maximal planes", technical report, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA
Manber U, 1989 *Introduction to Algorithms: A Creative Approach* (Addison-Wesley, Reading, MA)
Mantyla M, 1988 *An Introduction to Solid Modeling* (Computer Science Press, Rockville, MD)
Martin G E,1987 *Transformation Geometry: An Introduction to Symmetry* (Springer, New York)
Sedgewick R, 1988 *Algorithms* (Addison-Wesley, Reading, MA)
Stiny G, 1986, "A new line on drafting systems" *Design Computing* **1** 5-19
Stiny G, 1991, "The algebras of design" *Research in Engineering Design* **2** 171-181