1990

Generative geometric design and boundary solid grammars

Jeff A. Heisserman
Carnegie Mellon University

Carnegie Mellon University.Engineering Design Research Center.

Generative Geometric Design and Boundary Solid Grammars

by

Jeff Heisserman

EDRC 48-21-90

Generative Geometric Design and Boundary Solid Grammars

Thesis Proposal

by

Jeff A. Heisserman

Department of Architecture
and
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

This proposal introduces *boundary solid grammars*, a formalism for generating complex models of rigid solid objects. Solids are represented by their boundary elements, i.e. vertices, edges and faces, with coordinate geometry associated with the vertices. Labels may be associated with any of these elements. Rules match conditions of a solid or collection of solids and may modify them or create additional solids. A boundary solid grammar uses an initial solid and a set of rules to produce a language of solid models.

Unary operations are introduced to ensure the validity of the boundary representations. These operations take models that may have self-intersections, interpret the models considering the given geometry and face orientations, and produce valid models. The unary operations allow the use of boolean operations together with Euler, sweeping, tweaking and gluing operations in a unified, valid scheme.

The proposed formalism has been implemented. Grammars have been demonstrated that generate simple geometric forms including snowflakes, recursive octahedrons, "fractal" mountains, and spirals. Queen Anne houses have been characterized in a more extensive grammar.

This work has been supported by the Engineering Design Research Center, an NSF Engineering Research Center.

1 Introduction

Geometric design is central to the architectural and engineering disciplines. Practioners in these fields continually encounter geometric design problems. A few examples these problems are:

- Compose the rooms of a house according to a particular style and construct a massing model. From this model, construct the roof, floors, ceilings, and interior and exterior walls, and locate doors and windows.
- Given a volumetric model of the bays of a high-rise dffice building, place the structural columns, beams, shear walls and floor slabs.
- Given a model of a part, generate support structures for manufacturing the part using a stereo lithography process.
- Construct a model of a set of dies to manufacture a given part.
- Given a set of components for a computer, lay out the components and design aioasing to enclose them.

Current solid modeling systems provide users with little assistance in accomplishing these tasks. They generally provide operations to create, manipulate, and modify models. However, in order to construct the models previously mentioned, a user must examine the model, consider what parts should be added, and determine where and how to apply provided modeling operations to create the new components. Programming facilities are often added, however they provide little more than the ability to create designs with parametric variations.

This proposal introduces *boundary solid grammars*, a formalism for generating complex models of rigid solid objects. Solids are represented by their boundary elements, i.e. vertices, edges and faces, with coordinate geometry associated with the vertices. Labels may be associated with any of these elements. Rules match conditions of a solid or collection of solids and may modify them or create additional solids. A boundary solid grammar uses an initial solid and a set of rules to produce a language of solid models. The boundary representation, rules, and boundary solid grammar (BSG) formalism are presented in Sections 3 through 5.

Solid grammars use both local operations and global operations. However, local operations may cause a model to have self-intersections, thus be a source of geometric invalidity. *Unary operations* are introduced to ensure the validity of the boundary representations. These are global operations that take any topologically valid model and interpret the model, considering the given geometry and face orientations, to produce valid models. These operations are described in Section 6.

The proposed formalism has been implemented. Grammars have been demonstrated that generate simple geometric forms including snowflakes, recursive octahedrons, "fractal" mountains, and spirals. Queen Anne houses have been characterized in a more extensive grammar. A brief description of the "Genesis" boundary solid grammar interpreter and the demonstration grammars can be found in Section 7.

2 Related Work

Previous research related to boundary solid grammars can be grouped into the following areas: generative formalisms, solid modeling representation, operations and validity, and feature recognition.

The most influential work in the architectural domain is the shape grammar formalism [Sti80]. This formalism uses a representation of "shapes", or line segments, composing line drawings. Rewrite rules are defined by two shapes, the first, when matched, is removed and replaced by the second. Boundary solid grammars draw heavily on the ideas of shape grammars, while using a solid modeling representation and highly parametric rule matching and application.

Much of the application of shape grammars has been for composition and analysis in architectural design, including Palladian villas [SM78], Frank Lloyd Wright's Prairie Houses [KE81J, Queen Anne houses [Fle87, FCPG85], bungalows [DF81], modern Italian apartments (after the Casa Giuliani Frigerio) [Fle81], and Japanese tea houses [Kni81]. Additional shape grammars have been written to generate Chinese ice ray designs [Sti77], Moghul gardens [SM80], and chair-back designs [Kni80].

Graph grammars and L-systems, although not geometric representations, have been used to generate graphs or arrays that are mapped to polygons, lines, or primitive solids. Much of this work has been directed at describing biological growth, especially of plants and trees [PLH88, dREF+88].

Solid modeling representations have been investigated for several years now. Overviews of this work can be found in [BEH80, RV82, Req88]. [Req80] details the foundations of solid representations. A more recent presentation is given in [Hof89].

Local and boolean operations have been used previously in many solid modeling systems. Euler operations were introduced by Baumgart [Bau75] for use on his winged edge polyhedron representation. They have since been adopted by several research groups [ELS75, BHS80, Hil82, MS82]. The topological validity and completeness of Euler operations has been presented in [EW79, ManS4]. Euler operators have also been represented as graph productions [Fit87]. Early use of boolean operations on solid models are found in TIPS [OKK73] and PADL [V+74, VR77]. They are currently found in almost all solid modeling systems.

The quest for validity in boundary representations, and the development of the unary operations, have benefitted from the work demonstrating validity of constructive solid geometry (CSG) schemes and the boolean operations. Many of those ideas apply here to the unary operations. The unary operations also use methods for checking the validity of a boundary models [BEH80].

Checking match conditions for appling a solid grammar rule is equivalent to recognizing features of a solid. Some of the recent feature recognition research uses methods similar to those usod in boundary solid grammars. De Floriani used a face-based boundary representation and features described as topology graphs [DF89]. Pinilla used an edge-based representation augmented with arcs representing primitive geometric relations (parallel and perpendicular), and located subgraphs generated from a graph grammar description of features [PFP89]. The method of feature roco^nition presented here allows more flexible description of features and efficient implementation than previous methods.

3 Boundary Representation of Solids

A boundary solid grammar uses a boundary representation of rigid solid objects. The topology is represented as a graph composed of nodes and arcs, where the nodes are topological elements, and the arcs represent the adjacencies between elements. The geometry consists of vertex coordinates and plane equations for polyhedral solids. The topology and geometry together define a boundary representation solid model [Req80].

Our BSG uses a representation consisting of:

- **a *topology* graph with *vertex, edgehalf, loop, face, shell* and *solid nodes* corresponding to the split-edge data structure [Eas82], and *arcs* representing their adjacencies;**
- **sets of *design labels* associated with topology nodes, as desired; and**
- **coordinate *geometry* in Si^3 associated with each vertex, and a plane equation in ft^4 associated with each face.**

7

The split-edge data structure is a variation of the winged edge structure [Bau72]. It differs in that each edge is separated into two *edge-half* structures. One face and one vertex is associated with each edge-half, and each edge-half is associated with its other half.

Design labels distinguish topological elements of solids, and have several uses. Labels may be required as conditions in rules to restrict rule application. This is illustrated in an example grammar presented later in this proposal. Design labels may be used to reduce search by marking conditions needed in future operations. They also allow completed constructions of solids to terminate, while prohibiting termination of incomplete or invalid solids.

The graph notation used here, and graph operators and graph production definitions used later, are adapted from [LY78], which was originally introduced in [Nag76a, Nag76b].

Definition: A *b-graph* over the alphabet $E^* = E_n \cup E_a \cup EA \cup K^3 \cup R^4$ is a tuple $b = (K,(p_a)a£j:_\omega T,\backslash,'y,<f))$ where

1. **E_n = {VERTEX, EDGEHALF, LOOP, FACE,SHELL, SOLID} is the alphabet of topological element types;**
2. **E_a = *{vertex.eh, vcrtexj, edgeh.v, cw.eh, ccw.eh,* oMer.cA, *cdgchj, loop-v, loop.th, loopJ, faceJ, face^h, shellJ, skciUolid, solidjsheii)* specifies the alphabet of relations between topological elements;**
3. **EA is the alphabet of design labels;**
4. ***K* is a finite set of nodes;**
5. ***Pa Q K* X *K* are relations (arcs) over Jif, one for each *a* in E_a;**
6. ***T* : *K* —• E_n is a function from nodes to types;**
7. **A : *K* —• *Y* is a function from nodes to sets of design labels;**
8. **7 : K_v -» K^3 is a function that maps vertex nodes to vertex coordinates, where $K_v = \{v \mid v \in K \wedge j(v) = \text{VERTEX}\}$ is the set of vertex nodes; and**

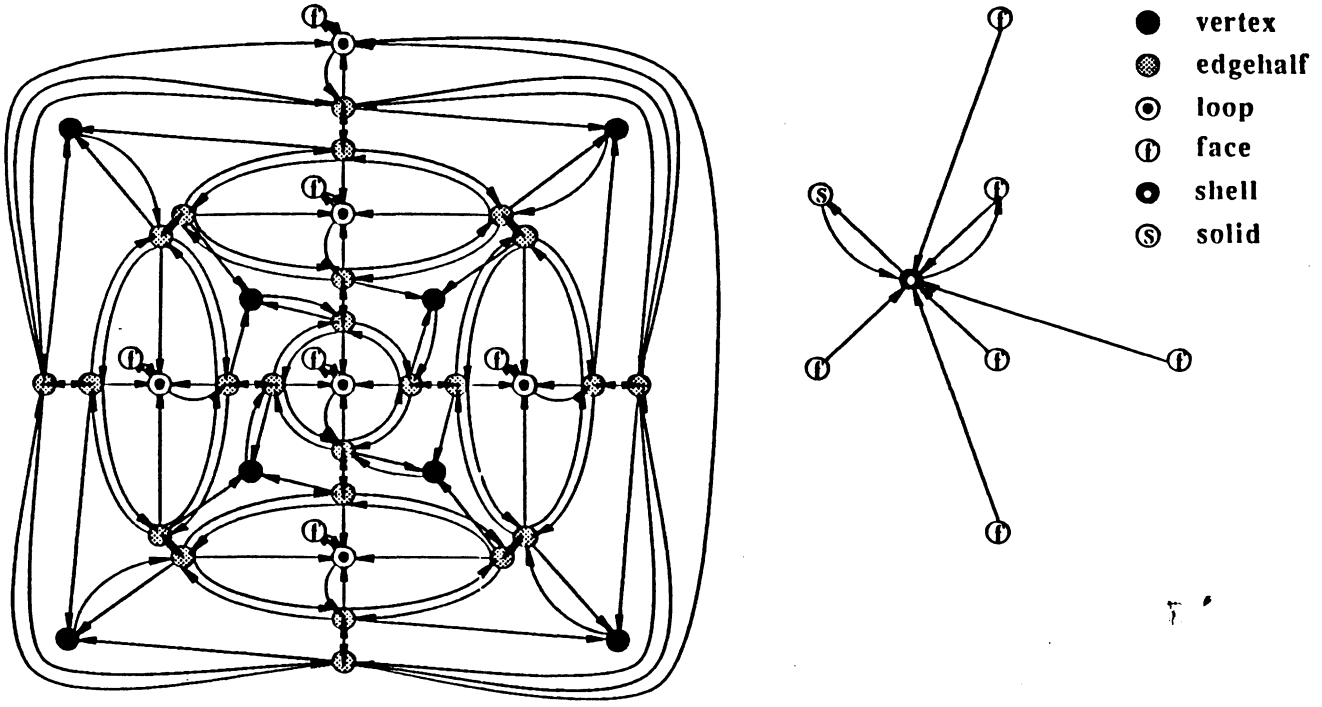


Figure 1: A cube represented with a b-graph.

9. $\phi : K_f \rightarrow \Re^4$ is a function that maps face nodes to face equations, where $K_f = \{f \mid f \in K \wedge \phi(f) = \text{FACE}\}$ is the set of face nodes.

Let $b(\Sigma_\star)$ denote the set of all b-graphs over the alphabet $\Sigma_\star$ and b_ϵ, the empty b-graph.

A b-graph can be interpreted as a directed graph with labeled nodes and arcs in the following sense: $(k_1, k_2) \in \rho_a$ means there is an arc from k_1 to k_2 of the type a; each node k is labeled with a type $\tau(k) \in \Sigma_n$ and a (possibly empty) set of design labels $\lambda(k)$; each vertex node $k_v \in K_v$ has an associated coordinate $\gamma(k_v)$; and each face node $k_f \in K_f$ has an associated face equations $\phi(k_f)$. Figure 1 illustrates the representation with the b-graph for a cube.

Not all b-graphs represent rigid solid objects. A b-graph may have nodes and arcs that do not correspond to a physical solid. For example, we can construct a b-graph with an "EDGEHALF" node connected to more than one "VERTEX" node with "*edgeh_v*" arcs. This is clearly invalid, since an edge connects exactly two vertices. A b-graph that is *topologically valid* must satisfy these conditions:

1. Every "VERTEX" is related to one "EDGEHALF" by an "*vertex_eh*" arc or one "LOOP" by an "*vertex_l*" arc (in the case of a loop with no edges).
2. Every "EDGEHALF" is related to exactly one "VERTEX" by an "*edgeh_v*" arc.
3. Every "EDGEHALF" is related with its other "EDGEHALF" by an "*other_eh*" arc.
4. Every "EDGEHALF" is related to exactly one "EDGEHALF" by a "*cw_eh*" arc.
5. Every "EDGEHALF" is related to exactly one "EDGEHALF" by a "*ccw_eh*" arc.

6. Every "EDGEHALF" is related to exactly one "LOOP" by an *"edgehj"* arc.

7. Every "LOOP" is related to one "EDGEHALF" by an "/oop.eA" arc or one "VERTEX" by an *"hop.v"* arc (in the case of a loop with no edges).

8. Every "LOOP" is related to exactly one "FACE" by an *"hopj"* arc.

9. Every "FACE" is related to exactly one "LOOP" by an *"facej"* arc.

10. Every "FACE" is related to exactly one "SHELL" by an *"facesh"* arc.

11. Every "SHELL" is related to exactly one "FACE" by an *"aheiu"* arc.

12. Every "SHELL" is related to exactly one "SOLID" by an *"sktiUolid"* arc.

13. Every "SOLID" is related to exactly one "SHELL" by an "*o/t(L«Ae//" arc.

These conditions are necessary, but not sufficient, to ensure topological validity. It is still possible to generate b-graphs that do not correspond to valid plane models and have faces with in&nsistent orientations. Euler operators are used in order to construct only topologically valid b-graphs.

The next section introduces the graph grammar notation and graph productions to implement the Euler operators on b-graphs.

A solid is *geometrically valid* if it satisfies these additional conditions [Man86]:

1. Faces of the solid may not intersect each other at their internal points.

2. Some edges of the solid may be entirely coincident, however their edge neighborhoods must be distinct.

3. Some edges may lie on a face. In this case, both faces adjacent to the edge must lie on the same side of the face.

4. Some vertices may lie on a face. In this case, all edges and faces adjacent to the vertex must lie on the same side of the face.

5. Some vertices may lie on an edge or on a vertex. In theses cases, all edges and faces adjacent to the vertex must lie on the same side of the surface defined by the faces of the incident edge or vertex.

A b-graph must be both topologically and geometrically valid in order to be a valid representation of a rigid solid object.

4 The Composition of Solid Rules

Solid rules are defined to allow abstract expression of complex match conditions and operators. This is accomplished using three levels of rules. Primitive match conditions correspond to matching in the b-graph. Primitive operations modify the b-graph with graph productions, and modifying the mappings of design labels and geometry. Logic rules or predicates are used to combine the primitive match conditions into higher ones, and to compose higher level operators from the primitive operations and conditions. Solid rules are composed of a set of match conditions that define where it is to be applied, and a series of operations that describe the modifications. We present the formulation of each of these three types of rules and their derivations in this section.

Primitive Matching and Operations

Primitive matching on b-graphs is a straightforward task of finding nodes and arcs, and accessing their type, labeling, and geometric functions. For example, we can find an edge-half e in a b-graph by finding a node of type EDGEHALF, that is r(e) = EDGEHALF. We can find the vertex vjLsfociated with that edge-half by finding a e<fyeA.varc in the b-graph from e to v. A label / on our edge-half can be found in the set of labels associated with that edge-half, / £ A(e). Finally, the coordinates of the vertex can be accessed with the geometric function 7, with (x, *y,z)* =: 7(1;).

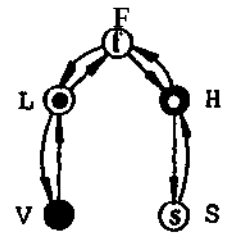
Labeling and geometric functions can be modified with primitive operations. Adding a label / to an element *e* is accomplished by modifying the labeling function A to A' such that A' = A(e) U {/}. Killing a label / modifies the labeling function A to A' such that A' = A(e) — {/}. Assigning coordinates c to a vertex *v* is equivalent to changing 7 to 7' such that 7'(i>) = c. Face equations are assigned with a similar modification to <*f*>. Element types and the type function *r* cannot be modified.

Primitive operations on the topological representation require more explanation. Euler operations modify boundary representation by adding and removing topological elements and relations, while maintaining topological validity. The Euler operations included here create minimal solids (MSSFLV), insert strut edges (MEV), split edges (ESPLIT), and split faces (MEFL). We present thoso Euler operations on b-graphs using graph productions.

A graph production has three parts: a subgraph that will be matched and removed from the graph being matched; a subgraph that will be inserted in place of the removed subgraph; and an embedding which specifies the connections between the inserted subgraph and the original graph. The embedding is specified using graph operators.

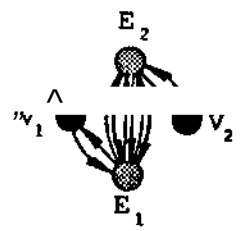
Definition: For the node set *K* of any b-graph with the node set K^1 of a given subgraph and any node *x* G *K*, the *graph operator A* yields a subset of *K*, written as *A(x)* according to the following recursive definition:

1. ***L{(x)* = *{y* E *K* — *K' A(y,x)* 6 *Pi)* specifies the set of nodes with i-labeled arcs that terminate in *x*.**
2. ***Ri(x)* = *{y e K - K'* A *(x, y) € pi}* denotes the set of nodes with i-labeled arcs leaving x.**
3. *AB(x)* = *{y* I *3z(z* G *B(x) A y e A{z)) Ay,z e K - K*^{1}*}* specifies the subset of nodes of *K* — *K'* **which are generated by sequencing graph operators.**



E: < *none* >

Figure 2: The MSSFLV graph production.



E: r_{edgeU} $= (E_1, E_3; LOOPL_{/OOP\text{-}V}(V_1))$

$r_{hoop.eh}$ $= (LOOPL\ loop\_b\ V_1); E_1)$

Figure 3: The MEV graph production when there are no edges in the loop.

4. ***(AUl))(x) = A(x)UB(x),***
specifies the subset of nodes of *K — K'* which are generated by the branching of operators.

Definition: A *graph production* is a triple *P* = (6/,6$_r$»*E)* with 6/,6$_r$ G &(£*) and an embedding transformation *E* = ((/a,r$_a$))$_{a€£_a}$, with ^ie embedding components /$_a$ and r$_a$ having the following form:

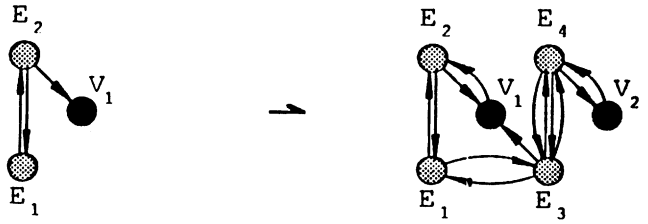
$$'. = \bigcup_{j=1}^{p} MK) \times \{*i)$$

$$r_a = \bigcup_{j=1}^{q} \{z_j\} \times A_j(y_j)$$

where *yj* G /0» ^j € AV, ^ij is an graph operator, and $p_y q \geq 1$. The b-graph 6/ is the left hand side and b_r the right hand side of the graph production. $Ki_y K_r$ are the sets of nodes in 6/$_{\%}$6$_r$, respectively. We may abbreviate *A(y)* x *{z}* by *(A(y);z)* and *{z}* x /l(y) by *(z;A(y)). (A;a,b)* is used to represent *(A;a)U (A;b)* with a similar notation for r-operators.

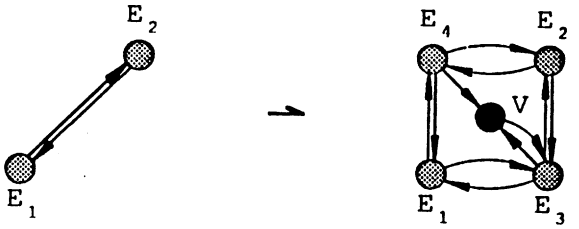
Informally, l_a specifies the a-labeled arcs from the existing graph to the newly inserted subgraph. *Aj(yj)* selects a subset of nodes in *K* - A'/, and connects these nodes to each of the nodes *{z$_3$}*. in the new subgraph with an a-labeled arc. Similarly, r_a specifies vhe a-labeled arcs from the subgraph to the existing graph.

Graph productions implementing Euler operators for the split edge data structure are presented in Figures 2 through 7. The node types in the embedding rules are redundant since the arcs usod are type specific, but the types are included here for clarity. These graph productions are used here



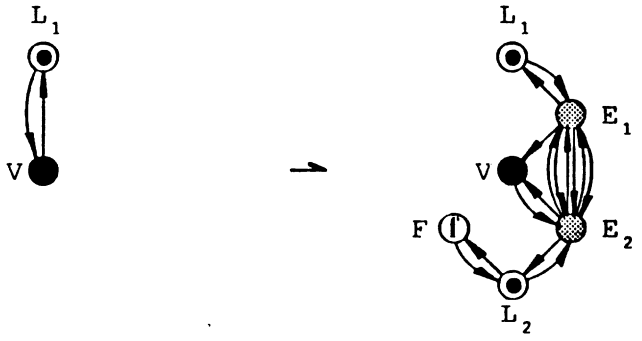
E: $r_{edgeh\_v} = (E_1; \text{VERTEX}R_{edgeh\_v}(E_1))$
$l_{cw\_eh} = (\text{EDGEHALF}L_{cw\_eh}(E_1); E_1)$
$l_{cw\_eh} = (\text{EDGEHALF}L_{cw\_eh}(E_2); E_2)$
$r_{cw\_eh} = (E_4; \text{EDGEHALF}R_{cw\_eh}(E_1))$
$r_{cw\_eh} = (E_2; \text{EDGEHALF}R_{cw\_eh}(E_2))$
$l_{ccw\_eh} = (\text{EDGEHALF}L_{ccw\_eh}(E_1); E_4)$
$l_{ccw\_eh} = (\text{EDGEHALF}L_{ccw\_eh}(E_2); E_2)$
$r_{ccw\_eh} = (E_1; \text{EDGEHALF}R_{ccw\_eh}(E_1))$
$r_{ccw\_eh} = (E_2; \text{EDGEHALF}R_{ccw\_eh}(E_2))$
$r_{edgeh\_l} = (E_1, E_3, E_4; \text{LOOP}R_{edgeh\_l}(E_1))$
$r_{edgeh\_l} = (E_2; \text{LOOP}R_{edgeh\_l}(E_2))$
$l_{loop\_eh} = (\text{LOOP}L_{loop\_eh}(E_1); E_1)$
$l_{loop\_eh} = (\text{LOOP}L_{loop\_eh}(E_2); E_2)$

Figure 4: The general MEV graph production.



E: $r_{edgeh\_v} = (E_1; \text{VERTEX}R_{edgeh\_v}(E_1))$
$r_{edgeh\_v} = (E_2; \text{VERTEX}R_{edgeh\_v}(E_2))$
$l_{cw\_eh} = (\text{EDGEHALF}L_{cw\_eh}(E_1); E_1)$
$l_{cw\_eh} = (\text{EDGEHALF}L_{cw\_eh}(E_2); E_2)$
$r_{cw\_eh} = (E_4; \text{EDGEHALF}R_{cw\_eh}(E_2))$
$r_{cw\_eh} = (E_3; \text{EDGEHALF}R_{cw\_eh}(E_1))$
$l_{ccw\_eh} = (\text{EDGEHALF}L_{ccw\_eh}(E_1); E_3)$
$l_{ccw\_eh} = (\text{EDGEHALF}L_{ccw\_eh}(E_2); E_4)$
$r_{ccw\_eh} = (E_1; \text{EDGEHALF}R_{ccw\_eh}(E_1))$
$r_{ccw\_eh} = (E_2; \text{EDGEHALF}R_{ccw\_eh}(E_2))$
$r_{edgeh\_l} = (E_1, E_3; \text{LOOP}R_{edgeh\_l}(E_1))$
$r_{edgeh\_l} = (E_2, E_4; \text{LOOP}R_{edgeh\_l}(E_2))$
$l_{loop\_eh} = (\text{LOOP}L_{loop\_eh}(E_1); E_1)$
$l_{loop\_eh} = (\text{LOOP}L_{loop\_eh}(E_2); E_2)$

Figure 5: The ESPLIT graph production.



E: $l_{face\_l} = (\text{FACE}L_{face\_l}(L_1); L_1)$
$r_{loop\_f} = (L_1; \text{FACE}R_{loop\_f}(L_1))$
$r_{face\_sh} = (F_1; \text{SHELL}R_{face\_sh}R_{loop\_f}(L_1))$

Figure 6: The MEFL graph production when there are no edges in the loop.

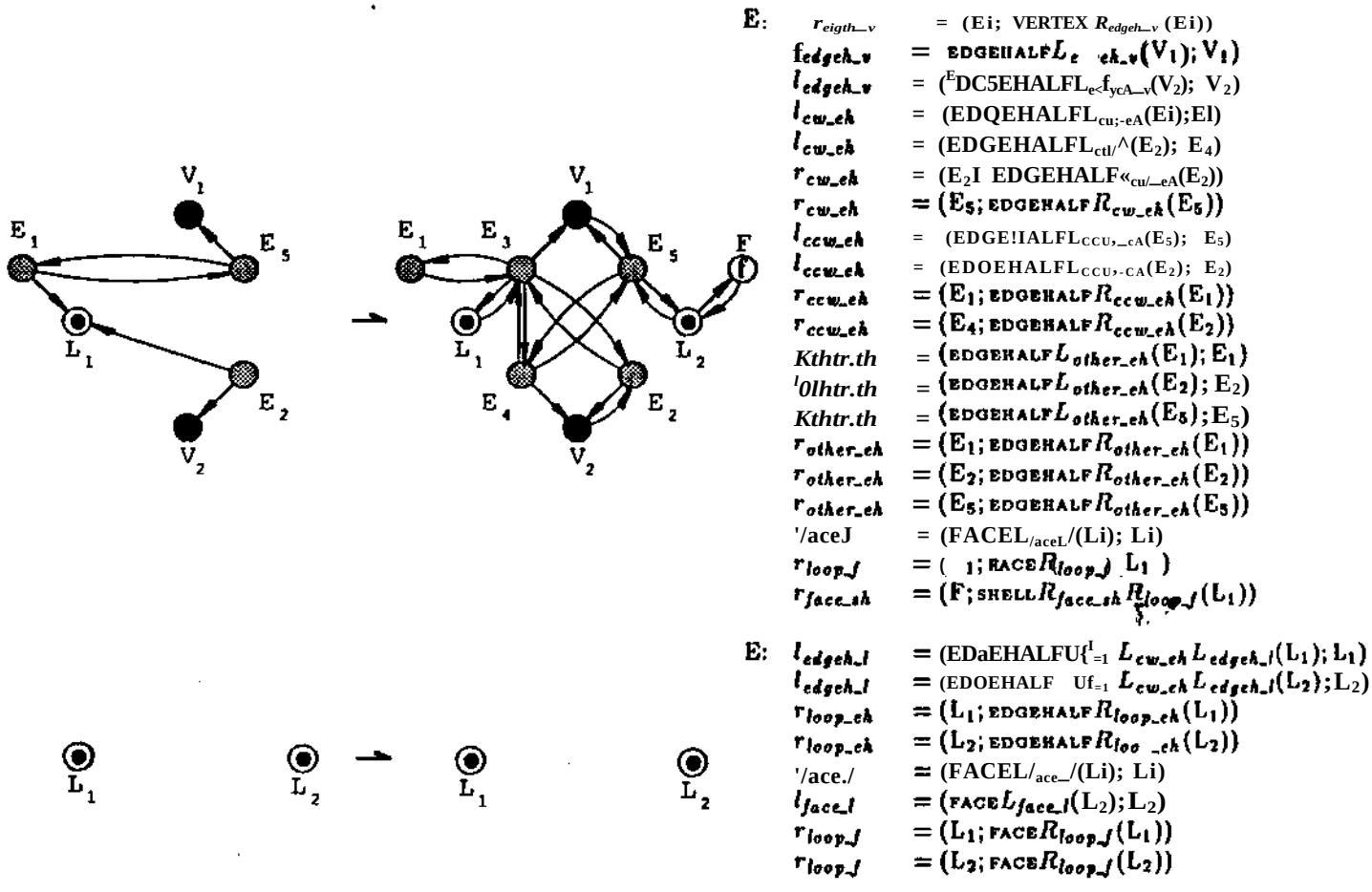


Figure 7: General MEHL as two sequentially applied graph productions.

as operators by binding key nodes of 6/ to specific nodes in a graph. Matching occurs first, followed by graph replacement and embedding.

Definition: A b-graph *V* € 6(E*) *is directly derivable* from another *b* £ 6(£*) by an operator *O* (abbreviated 6==>6') iff

1. 6/ Ç *b* and b_r Ç 6';

2. *b* - 6/ = 6' - 6_r; and

3. $In_a(b_r b')$ = $/_a$ and $Out_a(b_r b')$ = r_a for a G E_a

where $In_a(b_r b^f)$ is the set of all a-labeled incoming arcs originating in 6' - b_r and terminating in 6, and $Out_a(b_r b')$ is the set of all a-labeled outgoing arcs originating in b_r and terminating in *b* - 6_r.

Logical Composition of Primitives

We can construct complex matching conditions on b-graphs. In this section I explain how graph nodes, arcs and labels are represented as primitive predicates, and how to combine them with logic

rules into more useful and complex conditions.

Determining the type of topological elements may be represented as

$$t(*) = r(k) = t$$

for a node $k \in K$ and $t \in E_n$. For example, if a node k is of type VERTEX, then r(ifc) = VERTEX, and this is equivalent to the predicate vertex(fc). Similarly, relations between topological elements may be represented as

$$a(fci,\&_2) \equiv (*i>*2) \; e\,p_a$$

for nodes &i,A;$_2$ € *K*. Then the predicate eh_vertex(e,v) is equivalent to finding an "*edgehj*»" arc from *e* to *v* in p_e*dgch.v* Design labels may be represented as

$$label(e,/) \equiv / \in A(e).$$

Coordinates may be represented as

$$v.coord(r;,c) \equiv 7(1;) = c.$$

Face equations may be represented as

$$face.equation(/,e) = \phi(f) = e.$$

Predicates representing topological element types and relations, coordinates, and labels are considered *primitive conditions.*

Euler operations are specified as predicates that relate to the graph productions presented. For example,

$$mev(Vi,E_1,E_3,V_2)$$

is equivalent to the general mev graph production shown in Figure 3 with Vi, Ei, E3, V_2 bound to the corresponding key nodes. The simple mev graph production in Figure 2 is equivalent to

$$mev(Vi,nw//,Ei,V_2).$$

Similarly for

$$mssflv(S,II,F,L,V),$$

$$esplit(Ei,E_3,V),$$

$$mef1(V, null, V, null, E_3, L_2, F)$$

for the simple mef1 graph production, and

$$mefl(V_1,E_1,V_2,E_2,E_3,L_2,F)$$

for the general one. Element labels are created using

$$make\_label(e,/),$$

which is equivalent to changing A to A' such that A'(e) = A(e) U {/}, or

$$kill\_label(e,l)$$

which modifies A to A' such that A'(e) = A(e) - {/}. Vertex coordinates are assigned using

$$set\text{-}vertex(v,c),$$

which is equivalent to modifying 7 to 7' such that $Y(v) = c$. Similarly, face equations are modified with

set«face.equation(/, c),

which is equivalent to modifying <*f*> to <*f*>' such that <*f/{f)* = e. The predicates representing Euler operators, design label addition and removal, vertex coordinate assignment, and face equations assignment are *primitive operations.*

These primitive conditions and operations may be combined using logic rules. Predicates of the form

$$A:\text{-}B_u...,B_n.$$

can be constructed with predicates J3i,...,l?$_n$. For example, the implicit "other.v" relation, the relation between an edge-half and the vertex associated to its other edge-half, is a conjunction of the *"oiher.tk"* relation and the *"edgch^v"* relation:

```
other_v(Eh, Vertex):-
        other_eh(Eh, OtherEh),                                  ^ #
        edgeh_v(OtherEh_f Vertex).
```

Similarly, we express the length of an edge-half as the distance between the coordinates of the vertices associated with that edge-half and its other edge-half:

```
eh.length(Eh, Length):-
        edgeh_v(Eh, VI),
        other.v(Eh, V2),
        distance.v(Vl, V2, Length).
```

where distance_v(Vl, *V2,Length)* calculates the euclidean distance between the coordinates of two vertices *VI* and *V2.*

In this way, diverse graph matching conditions and conditions on the geometry can be combined to form higher level match conditions. Some of these match conditions include:

- the *lengths* of edges (as demonstrated), areas of faces, volumes and mass of solids;
- *angles* between edges and faces;
- *orientations* of faces and solids;
- *coincident, colinear* or *coplanar* vertices, edges and faces;
- the *centers* of edges, faces and solids;
- the *moment of inertia* of faces and solids;
- *intersections* of lines, planes, surfaces, edges, faces, and solids, including the intersection of two solids (boolean intersection) and *self-intersection of* a solid (unary intersection - described later in this proposal).

A single condition may be used to express the match conditions of an infinite set of graphs using recursive rules. Alternately, a condition may match greatly varied topology graphs using several predicates with the same head.

Logic rules define high level operators from primitive operators and match conditions. The match conditions allow an operator to respond to the context in which it is applied. For example, a single `extrude_face` operator extrudes any face of a solid, properly considering the number of edges in the face. Powerful operators, such as offsetting, boolean, and unary operators, may be constructed in this way. In addition, we can construct a single operator to be used in place of a set of rules. However, unlike match conditions, operators are required to succeed for any valid input.

Solid Rules

We have developed the methods for describing flexible matching conditions and useful operations. We use these conditions and operations to define solid rules.

Definition: A solid rule $\mathcal{R}$ is a pair $\alpha \longrightarrow \beta$, where

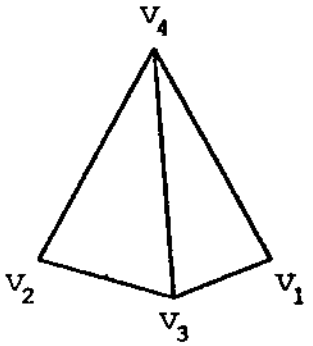
- $\alpha = C_1, \ldots, C_n$ for $C_i \in C$. C is a finite set of conditions of the form A :- $B_1, \ldots, B_n$ where each B_i is the head of one or more predicates in C or a primitive condition; and
- $\beta = O_1, \ldots, O_n$ is a sequence of operations, and each $O_i \in C$ or $O_i \in O$. O is a finite set of operations. Each operation in O is a sequence of conditions and operations of the form A :- $B_1, \ldots, B_n$ where $B_i \in O \cup C$ is a primitive condition, a primitive operation, or an operation in O. Each O_i is required to succeed for any valid input.

Solid rules derive one b-graph from another according to the following definition:

Definition: A b-graph $b' \in b(\Sigma_\star)$ is *directly derivable* from another $b \in b(\Sigma_\star)$ by a solid rule $\mathcal{R}$ (abbreviated $b \underset{\mathcal{R}}{\Longrightarrow} b'$) iff

1. all of the conditions $C_1, \ldots, C_n$ in $\alpha_\mathcal{R}$ are satisfied with respect to b;
2. for each of the operations $O_1, \ldots, O_n$ in $\beta_\mathcal{R}$
   - O_i is a condition and is satisfied with respect to b_{i-1} thus $b_{i-1} \underset{O_i}{\Longrightarrow} b_i$ and $b_{i-1} = b_i$, or
   - O_i is an operation and $b_{i-1} \underset{O_i}{\Longrightarrow} b_i$,

   and $b \underset{O_1}{\Longrightarrow} b_1 \underset{O_2}{\Longrightarrow} b_2 \ldots b_{n-1} \underset{O_n}{\Longrightarrow} b'$.



v_coord(t/i,(2,0,0)) label(f_{123}, a)
v_coord(t/2,(-l,√3,())) label(f_{124}, a)
v.coord(v3, (-1, -√3,0)) Iabel(/i3,|,a)
v.coord(t;4, (0,0,2^2)) label(/234,a)

Figure 8: An initial solid.

5 Boundary Solid Grammar Formalism

With the representation and solid rule definition in hand, we can now define a boundary solid grammar.

Definition: A *boundary solid grammar is* a tuple *Q* = (E*, TV,/,/?), where:

1. **E* as defined for b-graphs;**
2. **TV C EA is the set of non-terminal nodes;**
3. **/ £ *b(E+)* U $\{b_e\}$ is a topologically valid initial b-graph;**
4. ***R* is a finite set of solid rules.**

The initial solid is a topologically valid solid or collection of solids in the above representation. Modifications of the initial solid, and subsequent solids, are accomplished by the application of the set of solid rules.

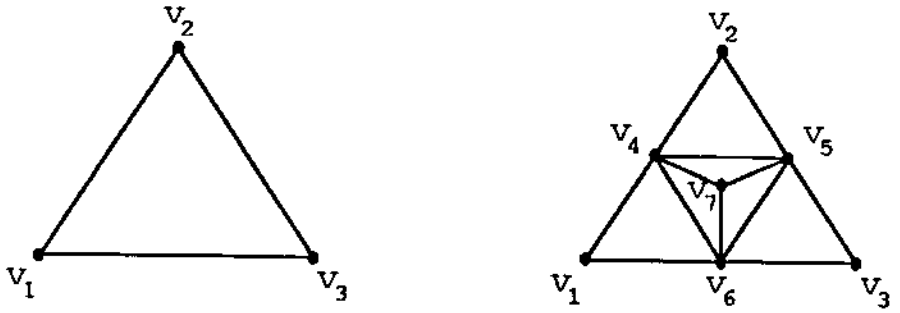
We can abbreviate *Q* = (E*,JV,J, *R)* to *Q* = *(E_{XY}NJ,R)* since E_n, E_a, $W^3$and ft^4 (of E*) are fixed for a given representation.

A boundary solid grammar *Q* generates a language *L(G)* of b-graphs. The BSG formalism is intended to provide a mechanism for generating languages that satisfy some design criteria, specified as constructive rules of design.

An Example Grammar

We can demonstrate the formalism with a simple grammar that generates a three dimensional analogue to Koch snowflakes. For the grammar *Q* = (E/,iV,/, *R),* E/ = {a} is the design label alphabet and none are non-terminals *(N* = 0). / is an initial model of the simplest snowHak<\ a tetrahedron (Figure 8). The four faces of the tetrahedron have "a" labels. *R* contains a single rule (shown in Figure 9). The rule matches on a face with an "a" label, a cycle of three edge-halves, and three vertices. It modifies the topology of the face by introducing four new vertices, nine edges, and five faces. These topoiogical modifications are represented by the application of Euler operators as follows:

- **Apply esplit three times, splitting each of the original edges and creating three new vertices.**



label($/_{123}$,a)

set_vertex(v_4, $(\gamma(v_1) + \gamma(v_2))$)
S6t_VGrtGx(t>5, $|(\gamma(v_1) + \gamma(v_3)))$
set.vertex(v_6, $\frac{1}{2}(\gamma(v_2) + \gamma(v_3))$)
SGt_VGrtGx(v_7, $\frac{1}{3}(7(\wedge 1) + 7(\wedge 2) + \gamma(v_3))$
$+\frac{1}{\sqrt{6}}$ $\gamma(v_2)$ - $l\{v\}$\ face_normal(/i23)

set_label(f_{146}, a)
set_label(f_{245}, a)
S6t-lab6l($/_{356}$,a)
set_label(f_{457}, a)
SGt.labGl($/_{567}$,a)
SGt_labGl($/_{467}$,a)

Figure 9: The example rule modifies a triangular face.

- **Apply mQf1 three times, once to each pair of newly created vertices. This divides the original face into four faces.**
- **Apply mQV to one of the new vertices to create the final (center) vertex.**
- **Apply mGf1 (twice) to the center vertex and each of the two new vertices not used in the last operation. This divides the center face into three faces.**

The rule finishes by calculating coordinates for the new vertices, and attaching labels to the now faces, facGJiormal calculates the surface normal of a face. A language of snowflakes is produced by this solid grammar that includes both regular and irregular examples. A portion of this language is shown in Figure 10. A more detailed snowllake, shown in Figure 11, was produced by the Conosis solid grammar interpreter.

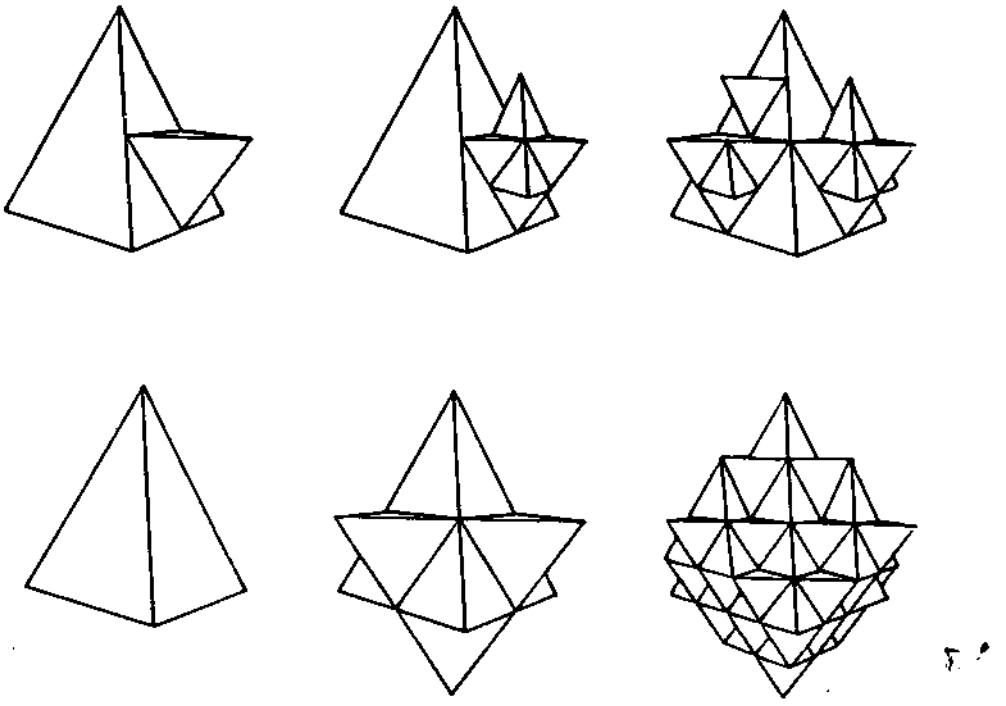


Figure 10: A portion of the language of snowflakes.

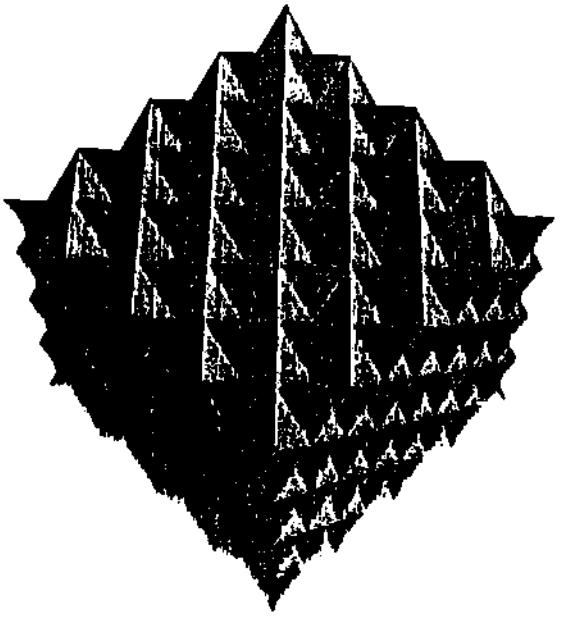


Figure 11: Another snowflake.

6 Unary Operations

The formalism described so far guarantees that the topologies constructed are valid, but allows arbitrary assignment of geometry. We can arbitrarily assign coordinates to vertices and construct models that have self-intersections and inconsistent face descriptions.

We would like an operation that takes any model with a valid topology and arbitrary geometry

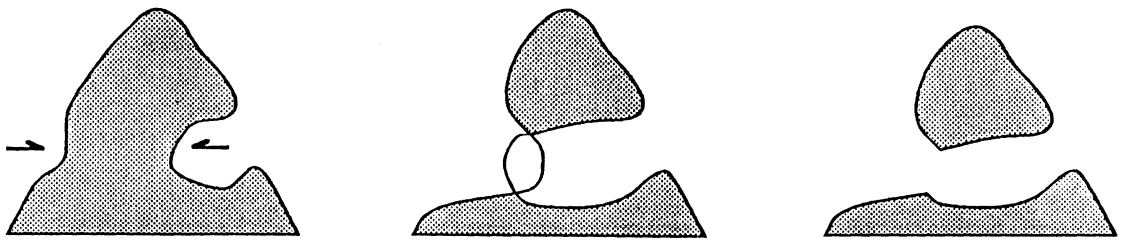


Figure 12: Stretching the boundaries of a solid inward.

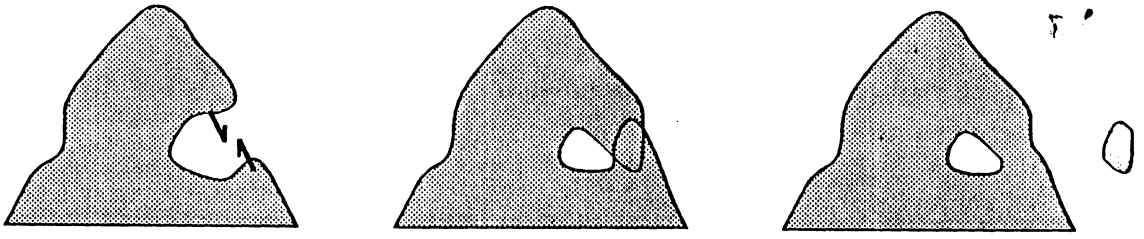


Figure 13: Stretching the boundaries of a solid outward.

and produces a model with a valid topology and geometry. In addition, the operation should leave a valid model unchanged. Changes introduced to an invalid model should correspond to our intuition of physical solids.

We can ask, "Why would a solid intersect itself?" One reason may be that two surfaces of a solid stretch inward enough that they overlap.[1] This is illustrated in Figure 12 . The volume between the two intersecting parts of the boundary is to the outside of both. Since the boundary separates the outside from itself, we really have a hole through the solid. The intersecting portions of the boundary are unwanted. Alternately, two parts of a solid may stretch outward far enough to intersect. We can see this in Figure 13 . Then we have parts of the boundary inside what we think of as solid. Since the boundary separates interior points of the inside, the intersecting portion of the boundary is unnecessary. It is also undesirable, since the computation of the volume based on the boundary would be incorrect with the intersecting elements.

The proposed *unary union* operator has the described characteristics. That is, the unary union of a solid (defined by its boundary) is the solid that contains all the points enclosed at least once by the boundary. The *unary intersection* of a solid is the solid that contains all the points enclosed more than once by the boundary. Unary union together with unary intersection comprise the unary operations.

We can define the unary operations more precisely using *winding* or *enclosing numbers*. A

[1]This example appears when constructing "fractal" mountains. Local modification of the boundary followed by a unary union operation allows caves to intersect and form holes in the mountain, and spires to intersect back into the mountain forming arches.

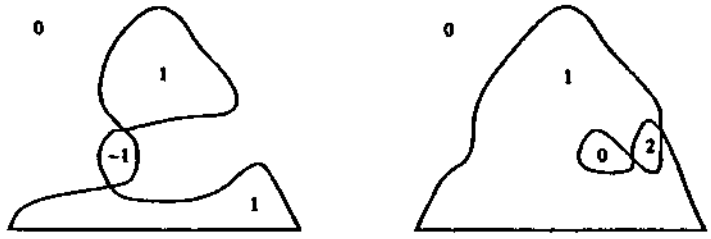


Figure 14: Classification of boundary elements using enclosing numbers.

winding number corresponds to the number of times a curve winds around a point not on the curve. The winding number of a point relative to a closed curve will always be an integer. Since we are working with (closed) 2-manifold surfaces, we will always have integer winding numbers. I will use the term enclosing number, which is more intuitive when our curves are enclosing surfaces in 3D [CS66]. The enclosing number of a point with respect to multiple 2-manifold sujffales is the sum of the enclosing numbers of that point with respect to each of the individual surfaces. This definition is necessary since we are representing multi-shell manifold solids.

We can now define the unary union of a solid as the solid defined by the boundary separating points with enclosing numbers zero or less from points with enclosing numbers one or greater. The unary intersection of a solid is the solid defined by the boundary separating points with enclosing numbers one or less from points with enclosing numbers two or greater. The generalized unary intersection of a solid 5, nJJ(5), is the solid defined by the boundary separating points with enclosing numbers n - 1 or less from points with enclosing numbers n or greater. The unary union is then $U_u(5) \equiv$ n£(5), and unary intersection is $n_u(S) \equiv$ *nl(S).*

Following the semantics outlined, any valid solid 5 will be identical to its unary union:

$$U_{tt}(S) = 5,$$

and the unary intersection of a valid solid is null:

$$\mathbf{n_u(5) = 0.}$$

Our hypothesis is that the unary operations are closed on boundary representations of r-sets [Req80]. The present representation is based on 2-manifolds, however the relaxed geometric conditions (of pseudo-manifolds [Man86]) described earlier allow implicit representation of r-sets. A set of generalized Euler operators for r-sets has been proposed [DS88] that should allow this work to be extended from 2-manifold to r-set representation. With the unary operators, we will bo able to build a system that is valid and constructs models using Euler, sweeping, tweaking and gluing operators, followed by unary operators (ensuring a valid model), then successively applying boolran operations or additional local and unary operations.

Initially, we propose the following algorithm for computation of the unary operations:

- Find all intersections between the faces, edges and vertices.

- **Modify the b-graph with Euler operators to represent these intersections explicitly in the topology.**
- **Separate the newly intersected shells.**
- **Calculate the enclosing numbers of points within each separated shell.**
- **Remove all unnecessary shells. For unary union, remove the shells not between the outside (points with enclosing numbers zero or less) and the inside (points with enclosing numbers one or greater). For unary intersection, remove all shells but those between singly enclosed points (with enclosing numbers one or less) and multiply enclosed points (with numbers two or greater).**

A more detailed algorithm is currently being developed and implemented, and will be presented in the thesis.

7 Research Plan

Work So Far

The Genesis boundary solid grammar interpreter has been under construction since November 1988. The first version was operational in January 1989. It was written in C-Prolog, and had facilities for boundary representation of solids, matching, local operations, rule representation and application, and wireframe display graphics. The boundary representation used the split-edge data structure with vertex, edge, face and solid elements. Local operations modified the representation via Euler operators (Msfv, Mev, Mef, Esplit) and coordinate geometry assignment.

Two grammars were constructed to run on the Version I interpreter. The first generated a 3D variant of the Koch snowflake. The second grammar generated 3-D volume layouts for Queen Anne houses, following Flemming's work [FCPG85, Fle87].

Development of the second version of the interpreter began in April 1989. The representation was extended to include loops and shells. The Euler operators were rewritten to use the new elements (Mssflv, Mev, Mefl, Esplit), and their operation was made to be consistent with the documentation for Vega II[2]. Many predicates were developed for extended solid rule matching and rule operations, allowing easier grammar construction. The implementation language was changed from C-Prolog to CLP(3?), a superset of prolog with constraints on real numbers, using the interpreter developed at Monash University. Surface rendering of models was added using HP Starbase graphics routines (in C).

Additional grammars were constructed to run on the Version II interpreter. Snowflake grammars generate both uniform and non-uniform snowflakes. An associated grammar generates uniform subdivisions of octahedrons. Solid models of "fractal" mountains are generated by a third grammar (a variant of the uniform snowflake grammar). A large variety of conch-like shells are generated by a spiral grammar. Finally, Queen Anne houses are generated with additional layout rules, room naming, roofs, room articulation, and porches.

What Remains To Be Done

The proposed third implementation will use a C-based CAD database (using parts of Vega II) integrated into the solid grammar interpreter to provide efficient access to the topological representation. This will provide a complete set of Euler operators as primitive operations. It will use the IBM CLP(3?) compiler in place of the Monash interpreter. An implementation of the unary operations will need to be added to the interpreter. We expect to implement additional calculations, for example area, volume, center of mass, and moment of inertia, that will allow for more interesting match conditions. Display routines will directly access the C based topological representation.

We propose to make several improvements to the demonstration grammars: additional detail on Queen Anne houses (construction of interior and exterior walls, porch wrapping, locate chimneys, roof detail, room layout based on areas); construct a grammar to automatically generate supports for models to be made by stereo lithography (sla); modify the existing grammars to operate within the new implementation - snowflakes, octahedrons, mountains, and spirals.

[2]Vega II is a solid modeling toolkit that was developed jointly by the Center of Art *k.* Technology and the Departments of Architecture and Design at Carnegie Mellon University. A programmer's reference manual [Lyo89j is available.

Figure 15: A house from the language of Queen Anne houses.

The unary union and unary intersection operations require complete description and precise algorithms for their computation. This algorithm will be demonstrated in an implementation. We intend to show that every topologically valid model can be correctly interpreted by the unary operations and will result in a valid model or models.

The BSG formalism has a form of the composition/decomposition problem found in shape grammars. The thesis will contain a presentation of the problem as it exists in BSG's and a solution within the formalism.

Finally, an analysis of the complexity of the rule matching algorithms will be presented in the thesis.

Schedule

- Jul 1990 - Complete interpreter with unary operations.
- Aug 1990 - Complete demonstration grammars and analysis.
- Dec 1990 - Complete thesis.

Expected Contributions

- Theory
  - The formal definition of boundary solid grammars.
  - The definition of unary operations on boundary representations.
  - A proof that topologically valid boundary models can be interpreted as valid boundary representations.

- A valid representation scheme for boundary representations using local operations, or a combination of local operations and boolean operations.

- Implementation
  - An implementation of a boundary solid grammar interpreter.
  - Algorithms for the computation of the unary operations.
  - An implementation of the unary operations.

- Application
  - A characterization of Queen Anne houses using boundary solid grammars.
  - Automated construction of supports for solid models being built using stereo lithography.

References

[Bau72] B. G. Baumgart. Winged-edge polyhedron representation. Technical Report £S-320, Stanford AI Laboratory, Stanford University, October 1972.

[Bau75] B. G. Baumgart. A polyhedron representation for computer vision. In *AFIPS Conf. Proc,* volume 44, pages 589-596, 1975.

[BEH80] A. Baer, C. Eastman, and M. Henrion. Geometric modelling: A survey. Technical Report IBS Research Report No. 4, Department of Architecture, Carnegie Mellon University, February 1980.

[BHS80] I. C. Braid, R. C. Hillyard, and I. A. Stroud. Stepwise construction of polyhedra in geometric modeling. In K. W. Brodlie, editor, *Mathematical Methods in Computer Graphics and Design,* pages 123,141. Academic Press, New York, 1980.

[CS66] W. G. Chinn and N. E. Steenrod. *First Concepts of Topology.* Random House, New York, 19G6.

[DF81] F. Downing and U. Flemming. The bungalows of Buffalo. *Environment and Planning D,* 8:269-293, 1981.

[DF89] Leila De Floriani. Feature extraction from boundary models of three-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* ll(8):785-798, 1989.

[dREF+88] Philippe de Reffye, Claude Edelin, Jean Francon, Marc Jaeger, and Claude Puech. Plant models faithful to botanical structure and development. *ACM Computer Graphics,* 22(4):151-158, 1988.

[DS88] II. Desaulniers and N. F. Stewart. Generalized Euler operators for r-sets. Technical Report 649, Department d'Informatique et de Recherche Operationnelle, Universite de Montreal, April 1988.

[Eas82] C. M. Eastman. Introduction to computer aided design. Technical Report Course Notes, Carnegie Mellon University, 1982.

[ELS75] C. Eastman, J. Lividini, and D. Stoker. A database for designing large physical systems. In *Proc. 1975 National Computer Conference,* pages 603-611. AFIPS Press, New Jersey, 1975.

[EW79] C. M. Eastman and K. Weiler. Geometric modeling using the euler operators. In *Proc. First Ann. Conf. Computer Graphics and CAD/CAM Systems,* pages 248-254, Cambridge, Mass., April 1979. M.I.T. Press.

[FCPG85] U. Flemming, Robert Coyne, Shakunthala Pithavadian, and Raymond Gindroz. A pattern book for Shadyside. Technical report, Department of Architecture, Carnegie Mellon University, December 1985.

[Fit87] Patrick Fitzhorn. A linguistic formalism for engineering solid modeling. In *Graph-Grammars and Their Application to Computer Science,* pages 202-215, Berlin, 1987. Springer-Verlag.

[Fle8l] U. Flemming. The secret of the Casa Guiliani Frigerio. *Environment and Planning B,* 8:87-96, 1981.

[Fle87] U. Flemming. More than the sum of parts: the grammar of Queen Anne houses. *Environment and Planning B,* 14:323-350, 1987.

[Hil82] Robin Hillyard. The build group of solid modelers. *IEEE Computer Graphics and Applications,* 2(2):43-52, March 1982.

[Hof89] Christoph M. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufmann Publishers, San Mateo, 1989.

[KE81] II. Koning and J. Eizenberg. The language of the prairie: Frank Lloyd Wright's prairie houses. *Environment and Planning B,* 8:295-323, 1981.

[Kni80] T. W. Knight. The generation of llepplewhite-style chair-back designs. *Environment and Planning B,* 7:227-238, 1980.

[Kni81] T. W. Knight. The forty-one steps. *Environment and Planning B,* 8:97-114, 1981.

[LY78] L. S. Levy and Kang Yueh. On labelled graph grammars. *Computing,* 20:109-125, 1978.

[Lyo89] Elizabeth Lyons. Vegall programmer's reference guide. Technical report, Center for Art and Technology, Carnegie Mellon University, May 1989.

[Man84] Martii Mantyla. A note on the modeling space of Euler operators. *Computer Vision, Graphics, and Image Processing,* 2G(l):45-60, April 1984.

[Man86] Martii Mantyla. Boolean operations of 2-manifolds through vertex neighborhood classification. *ACM Transactions on Graphics,* 5(l):l-29, January 1986.

[MS82] M. Mantyla and R. Sulonen. GWB: A solid modeler with Euler operators. *IEEE Computer Graphics and Applications,* 2(7):17-31, September 1982.

[Nag76a] M. Nagl. Formal languages of labelled graphs. *Computing,* 16:113-137, 1976.

[Nag76b] M. Nagl. Graph rewriting systems and their application in biology. In *Lecture Notes in Biomathematics,* volume 11, pages 135-156, Berlin, 1976. Springer-Verlag.

[OKK73] N. Okino, Y. Kakazu, and II. Kubo. Tips-1: Technical information processing system for computer aided design and manufacturing. In J. Hatvany, editor, *Computer Languages for Numerical Control,* pages 141-150. North Holland, Amsterdam, 1973.

[PFP89] J. M. Pinilla, S. Finger, and F. B. Prinz. Shape feature description and recognition using an augmented topology graph grammar. In *NSF Engineering Design Research Conference*, pages 285-300. University of Massachusetts, Amherst MA, June 11-14 1989.

[PLH88] Przemyslaw Prusinkiewicz, Aristid Lindemayer, and James Hanan. Developmental models of herbateous plants for computer imagery purposes. *ACM Computer Graphics*, 22(4):141-150, 1988.

[Req80] A. A. G. Requicha. Representation of rigid solids: Theory, methods, and systems. *ACM Computing Surveys*, 12(4):437-464, 1980.

[Req88] A. A. G. Requicha. Solid modeling - a 1988 update. In B. Ravani, editor, *CAD Based Programming for Sensory Robots*, pages 3-22. Springer Verlag, New York, 1988.

[RV82] A. A. G. Requicha and II. B. Voelcker. Solid modeling: A historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 2(2):9-24, March 1982.

[SM78] George Stiny and William J. Mitchell. The Pailadian grammar. *Environment and Planning B*, 5:5-18, 1978.

[SM80] George Stiny and William J. Mitchell. The grammar of paradise: on the generation of Mughul gardens. *Environment and Planning B*, 7:209-226, 1980.

[Sti77] George Stiny. Ice-ray: a note on the generaation of Chinese lattice designs. *Environment and Planning B*, 4:89-98, 1977.

[Sti80] George Stiny. Introduction to shape and shape grammars. *Environment and Planning B*, 7:343-351, 1980.

[V+74] II. B. Voelcker et al. An introduction to PADL: Characteristics, status, and rationale. Technical Report Tech. Memo No. 22, Production Automation Project, University of Rochester, 1974.

[VR77] II. B. Voelcker and A. A. G. Requicha. Geometric modeling of physical parts and processes. *IEEE Computer*, 10:48-57, 1977.