Relational Specification of Surface Subdivision Algorithms

Colin Smith, Przemyslaw Prusinkiewicz, and Faramarz Samavati
Department of Computer Science
University of Calgary

Abstract

Many polygon mesh algorithms operate in a local manner, yet are formally specified using global indexing schemes. We address this discrepancy by defining a set of local operations on polygon meshes in relational, index-free terms. We also introduce the vv programming language to express these operations in a machine-readable form. We then apply vv to specify several surface subdivision algorithms. These specifications can be directly executed by the corresponding modeling software.

1 Introduction

Ideally, a problem description should clearly reflect its nature. The data should be organized to reflect the relations inherent in the problem, and the language used to describe the solution should focus on its essence. Superfluous elements should be avoided, as they obfuscate the nature of the problem and its solution. When the data and relations are elegantly organized, the solution often becomes simple and easy to understand and implement.

The particular problem that we address in this paper is that of dealing with local properties and local transformations of polygon meshes. Locality is one of the most fundamental characteristics of systems. It means that: (a) a neighborhood relation is defined on the elements of the system, and (b) each element of the system changes its state according to its own state and the state of its neighbors, to the exclusion the elements positioned farther away. The search for, and study of, local mechanisms that underpin observed phenomena has been one of the central and fruitful directions in natural sciences, from physics to biology.

Many computer graphics algorithms also have local character. A good example is given by subdivision algorithms in geometric modeling. Their locality is intuitively captured when subdivision algorithms are described in terms of *masks* [31] (also referred to as *stencils* [27]). The prevalent formal definitions of subdivision algorithms, however, do not take advantage of the simplicity and locality of the masks, but rely on a global enumeration (indexing) of the polygon mesh elements. The appeal of indices is that they are a standard mathematical notation, and are closely coupled with the array data structures supported by most programming languages. Unfortunately, they are deficient in several respects:

- The use of indices does not adhere to the philosophy of describing local processes in local terms. For example, one can write the expressions $i - 1$ or $i + 1$, which refer to the immediate neighbors in a linear structure, as easily as $i - 100$ or $2i$, which do not.
- The indexed elements are often not arranged into a regular grid. This complicates the indexing scheme and index arithmetic, and makes them error-prone.
- The indexed elements must be dynamically renumbered as their number and arrangement change.
- Indexed notation has questionable value from the viewpoint of algorithm implementation, which may use a different indexing scheme than that used to specify the algorithm, or rely on pointers rather than index arithmetic when identifying neighbors.
- Index-heavy notation is hard to read.

We address these deficiencies by introducing the *vertex-vertex polygon mesh representation*, and the corresponding set of operations, the *vertex-vertex algebra*, which make it possible to describe local operations on polygon meshes in relational terms. This means that we identify elements of the structure with respect to each other, avoiding absolute identifiers such as coordinates or indices. We also introduce vv, an extension of the C++ programming language, for expressing these operations in a machine-readable form. This results in the *language + engine* modeling paradigm, which simplifies the implementation of individual algorithms by treating them as input to a multi-purpose modeling program. We demonstrate the usefulness of this paradigm by presenting concise vv specifications of several subdivision algorithms.

2 Background

Local modifications to structured objects are the essence of development. Consequently, the problem of referring to the neighbors often occurs in biologically-motivated models of computation. In cellular automata, for example, the neighbors of a given cell may be specified using index arithmetic, or in a relational manner, using the directions north, south, east and west. Giavitto and Michel [10] explored the advantages of the relational approach and generalized it to arbitrary regular tessellations (group-based fields). They have also proposed a programming language to capture locally defined structures that, unlike cellular automata and group-based-fields, may grow and dynamically reconfigure [11].

Growing geometric structures with linear and branching topology, have been constructed since the late 1960s as models of multicellular organisms, in particular plants. The relational approach to the identification of the elements of these structures is exemplified by the formalism of L-systems [19, 24]. A structure is represented by a sequence of symbols. This sequential arrangement automatically determines the neighborhood relations between the elements.

L-systems with affine geometry interpretation have recently been shown to provide a compact formal specifications of subdivision algorithms for curves [25]. For example, Chaikin's corner-cutting algorithm [6] is given by the production:

$$P(v_l) < P(v) > P(v_r) \rightarrow P\left(\tfrac{1}{4}v_l + \tfrac{3}{4}v\right) P\left(\tfrac{3}{4}v + \tfrac{1}{4}v_r\right). \qquad (1)$$

Here $P(x)$ denotes a point at location x; symbols $<$, $>$ and $\rightarrow$ separate the left context, the strict predecessor, the right context, and the successor of the production; and the arithmetic operators specify the affine combinations of the argument points. In a process akin to cell division in a developing organism, this production replaces the parent point by two descendant points, with the locations dependent on the context (Figure 1).

The simplicity, clarity, and compactness of the L-system specification of subdivision curves, combined with the possibility of executing them using an L-system-based modeling software [25], have motivated our quest for an extension that could be applied to polygon meshes as well. Unfortunately, the existing grammar-based formalisms fall short of this goal. *Map L-systems* [20, 24] can generate the topology of some polygon meshes, but do not offer flexible control over the resulting geometry and are difficult to specify. Similarly, *graph grammars* [26] extend formal grammars from

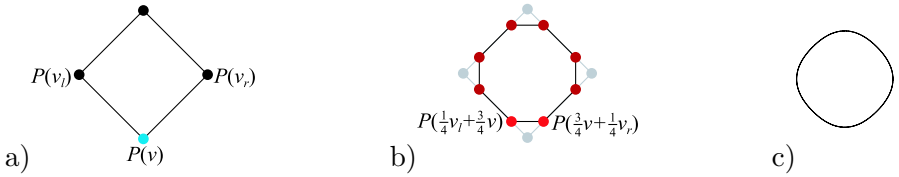


Figure 1: Chaikin subdivision process described by an L-system production (Equation 1). a) The initial polygon. Labels refer to an arbitrarily chosen point $P(v)$. b) The result of the first iteration of the algorithm. c) The curve after several subdivision steps.

strings to graphs. However, the development of graph theory has been focused on the context-free case, and more general formulations are difficult to use.

We attribute the deficiencies of graph grammars to the loss of information that occurs during production application. The predecessor is removed from the structure before the successor is inserted into it, and thus details of the predecessor's connections are not available for reconnecting the successor. To address this problem, we propose to operate in a more gradual manner, possibly adding new nodes and edges before the old ones have been removed. The old elements may thus serve as a scaffolding for introducing the new ones. This technique preserves the purely local operation of the grammar-based approaches, but departs from their declarative character, because modifications to structures are now specified as sequences of imperative operations.

The ease of performing local operations on polygon meshes depends on the mesh representation, which should be conducive to relational information gathering and mesh transformations. Well known examples of such representations include the *winged-edge* [4], and *quad-edge* [12] representations. Pursuing objectives closer to ours, Egli and Stewart [9] applied *cellular complexes* [23] to specify Catmull-Clark [5] subdivision in a relational manner. Lienhardt [18] showed that local operations involved in subdivision algorithms can be defined using *G-maps* [16, 17]. More recently, Velho [29] developed a method for describing subdivision algorithms using stellar operators [15] that act on a *half-edge* structure [22].

We have selected yet another representation, based on the mathematical notion of *graph rotation systems* [8, 30]. A rotation system associates each vertex of a polygon mesh with an oriented circular list of its neighboring vertices. A set of these lists, defined for each vertex, completely represents

the topology of a 2-manifold mesh [30]. Graph rotation systems have been introduced to computer graphics by Akleman, Chen and Srinivasan [1, 2, 3] as a formal basis for the *doubly linked face list* representation of 2-manifold meshes. Akleman et al. have also defined a set of operations on this representation, which they used to implement interactive polygon mesh modeling tools. Below we introduce *vertex-vertex systems* as a different data structure related to the graph rotation systems. It makes it possible to implement a set of graph manipulation operations in an intuitive and efficient manner.

3 Vertex-vertex systems

3.1 Definitions

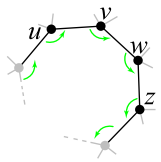


Figure 2: A polygon identification in a graph rotation system.

Let U be an enumerable set, or the *universe*, of elements called *abstract vertices*. We assume that U is ordered by a relation $<$; this assumption simplifies the implementation of many algorithms (Section 4). Next, let $N : U \mapsto 2^U$ be a function that takes every vertex $v \in U$ to a finite subset $v^\star \subset U$ of other vertices ($v \notin v^\star$). We call the set $v^\star$ the *neighborhood*, and its elements the *neighbors*[1] of v. Finally, let the *vertex set* $S \subset U$ be a finite subset of the universe U, and N_S be the restriction of the neighborhood function N to the domain S; thus $N_S(v) = v^\star$ if $N(v) = v^\star$ and $v \in S$ (the elements of $v^\star$ may lay outside S). We call the pair $\langle S, N_S \rangle$ a *vertex-vertex structure* over the set S with neighborhood N_S.

An *undirected graph* over a vertex set S is a vertex-vertex structure over S, in which: (a) all neighborhoods are included in S (the vertex set S is *closed* with respect to the function N), and (b) vertex u is in the neighborhood of v if an only if vertex v is in the neighborhood of u ($u \in v^\star$ if and only if $u \in v^\star$, the *symmetry condition*). The pairs (u, v) of vertices that are in the neighborhood of each other are called *edges* of the graph. An edge is *oriented* if the pair (u, v) is considered different from (v, u).

A *vertex-vertex rotation system*, or *vertex-vertex system* for short, is a vertex-vertex structure in which the vertices in each neighborhood form a cyclic permutation (i.e., are arranged into a circular list). A *graph rotation system* is a vertex-vertex system that is both a graph and a vertex-vertex

[1] Our terminology is motivated by the practice of referring to adjacent cells in a grid as neighbors. It should not be confused with the definition of neighborhood in topology.

rotation system.

A *polygon mesh* is a collections of vertices, edges bound by vertex pairs, and polygons bound by sequences of edges and vertices. A mesh is a *closed 2-manifold* if it is everywhere locally homeomorphic to an open disk, and a *2-manifold with boubdary* if it is everywhere locally homeomorphic to an open disk or half-disk. A manifold is *orientable* if it has two sides [30].

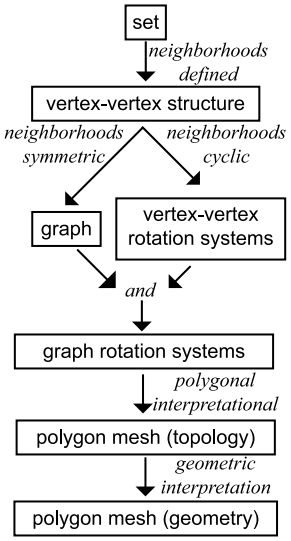


Figure 3: Relations between notions pertinent to vertex-vertex systems

A *polygonal interpretation* of a vertex-vertex system maps it into a polygon mesh. The interpretations that we consider in this paper are variants of the *Edmonds' permutation technique* [8, 30, 2], which is defined for connected graph rotation systems. It defines polygons of the mesh using the following algorithm (Figure 2). Given an oriented edge (u, v) in S, we find the oriented edge (v, w) such that w immediately follows u in the *cyclic* neighborhood of v. Next, we find the oriented edge (w, z) such that z immediately follows v in the neighborhood of w. We continue this process until we return to the starting point u. The resulting *orbit* (cyclic permutation) of vertices $u, v, w, z, \ldots$ and the edges that connect them are the boundaries of a polygon. By considering all such orbits in S, we obtain a polygon mesh with polygons on both sides of each (unoriented) edge. From this construction it immediately follows that the resulting mesh is a uniquely defined, orientable, closed 2-manifold (see [30] for a formal proof).

A function f defined on a vertex set S assigns a *property* $f(v)$ to each vertex $v \in S$. In addition to the neighborhoods defined above, vertex properties may include for example a label (drawn from a finite or an infinite set), position, normal vector, and color.

Vertex *positions* are a crucial aspect of the *geometric interpretation* of vertex-vertex systems. We will consider geometric interpretations in which edges are drawn as straight lines between vertices, and polygons are properly defined if their vertices and edges are coplanar.

The above progression of notions is summarized in Figure 3. It suggests that polygon meshes can be manipulated using operations defined on sets (set-theoretic operations), vertex-vertex systems and graphs (topological operations), and polygon meshes (geometric operations). The crucial problem is the manipulation of topology. We address it by introducing a set of op-

erations that modify at most one neighborhood at a time, and transform a vertex-vertex system into another vertex-vertex system. The individual operations do not necessarily transform graphs into graphs, because they may create *incomplete neighbors* that violate the symmetry condition ($u \in v^\star$ but $v \notin u^\star$).

3.2 The vertex-vertex algebra

The *vertex-vertex algebra* is the class of vertex-vertex rotation systems with a set of operations defined on them. We introduce these operations using a mathematical notation that combines standard and new mathematical symbols. We also present the equivalent expressions and statements of the vv language. A further description of this language and its implementation is given in Section 3.3.

3.2.1 Set-theoretic operations

In the vv language, vertex sets are a predefined data type. A set S is created using the declaration mesh S, and is in existence according to the standard scoping rules of C++. The vv language supports a subset of the standard set operations, listed in Table 1. In addition to operations that return a set as the result, vv includes iteration operators for flow control in vv programs.

Name	Math. notation	vv statement
set creation	let $S \subset U$	mesh S
assignment	$S = T$	$S = T$
union	$S = S \cup T$	merge S with T
addition of an element	$S = S \cup \{v\}$	add v to S
removal of an element	$S = S - \{v\}$	remove v from S
iteration over a set	$\forall v \in S$	forall v in S
iteration over neighbors	$\forall x \in v^\star$	forall x in v

Table 1: Set-theoretic operations supported by the vv language

3.2.2 Topological operations

Topological operations are the core of the vertex-vertex algebra. They are divided into three groups: *query*, *selection*, and *editing* operations. Query operations return information about vertices. Selection operations return an element of a vertex neighborhood. Editing operations modify a vertex-

vertex system. Definition of these operations are given in Table 2. The last column in this table refer to the illustrations in Figure 4.

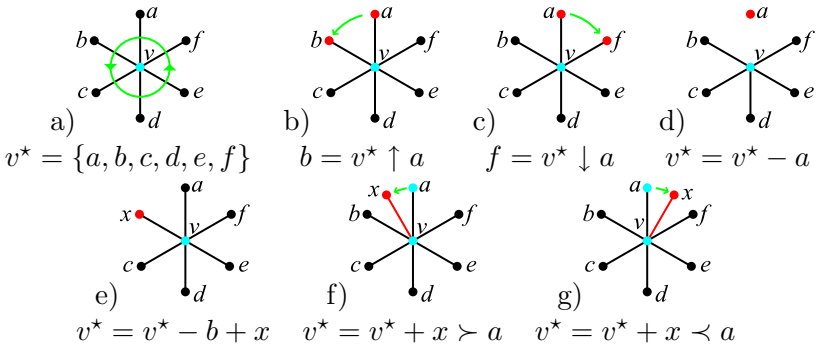


Figure 4: Examples of operations in the vertex-vertex algebra. a) Setting the initial neighborhood of vertex v. b-g) The results of selection and editing operations applied to v.

3.2.3 Geometric operations

We use the standard functional notation $f(v)$ or vv expression v\$f to associate property f with a vertex v. A special case is the position of a vertex, denoted $\overline{v}$ or v\$pos. Positions can be assigned explicitly, by referring to an underlying coordinate system, or result from affine geometry combinations and vector operations applied to the previously defined points. We use the standard C++ operator overloading mechanism to extend arithmetic operators to positions and vectors.

3.2.4 Coordination operations

Operations of the vertex-vertex algebra are commonly iterated over vertex sets. This raises important questions concerning the sequencing of these individual operations. For example, if the same operation is to be performed on a pair of neighboring vertices u and v, the results may be different depending on whether u is modified first, v is modified first, or both vertices are modified simultaneously. To eliminate the unwanted dependence on the execution sequence, we introduce the *coordination operation* synchronize S, which creates a copy $'v$ of each vertex v in the set S. All subsequent operations on the vertices $v \in S$ (until the next *synchronize* statement) do not affect the vertices $'v$, which continue to store the "old" values of vertex

Name	Math. notation	vv statement	Description	Note	Fig
		Query operations			
membership	$v \in x^\star$	is x in v	true iff vertex x is in the neighborhood of v		
order	$x < v$	$x < v$	true iff vertex x precedes vertex v in the universe U		
valence	$\|v^\star\|$	valence v	returns the number of neighbors of vertex v		
		Selection operations			
any	let $v \in x^\star$	any in v	returns a random neighbor of v	1	
next	$v^\star \uparrow x$	nextto x in v	returns vertex that follows x in the neighborhood of v	2	b
previous	$v^\star \downarrow x$	prevto x in v	returns vertex that precedes x in the neighborhood of v	2	c
		Editing operations			
create	let $v \in U$	vertex v	create vertex		
set neighborhood	$v^\star = \{a, b, c\}$	make $\{a, b, c\}$ nb_of v	set the neighborhood of v to the given circular list	3	a
erase	$v^\star = v^\star - x$	erase x from v	remove x from the neighborhood of v if $v \in x^\star$	4	d
replace	$v^\star = v^\star - a + x$	replace a with x in v	substitute x for a in the neighborhood of v	5	e
splice after	$v^\star + x \succ a$	splice x after a in v	insert x immediately after a in the neighborhood of v	5	f
splice before	$v^\star + x \prec a$	splice x before a in v	insert x immediately before a in the neighborhood of v	5	g

1) returns the null vertex if $v^\star$ is empty.
2) returns the null vertex if $x \notin v^\star$.
3) not defined (error reported) if v appears in the list, or the same vertex in listed twice.
4) no effect if $v \notin x^\star$.
5) no effect if $v \notin a^\star$; not defined (error reported) if $x = v$ or $v \in x^\star$.

Table 2: Topological operations of the vertex-vertex algebra

attributes. For example, '$v$$pos denotes the position of vertex v at the time when the *synchronize* statement was last issued, whereas $v$$pos denotes the current position of v. Similarly, '$v^\star$ and $v^\star$ denote the old and current neighborhoods of v. The use of old attributes instead of the current ones makes it possible to iterate over the elements of a set in any order without affecting the iteration results.

3.3 Implementation of vertex-vertex systems

The software implementation of vertex-vertex systems is a set of programs and libraries collectively called the vv environment. The central component of this environment is vvlib, a C++ library containing data structures and functions implementing the vertex-vertex polygon mesh representation and algebra. The user can refer to these structures and functions directly from a program written in C++, or from a program written in the vv language.

The vv language extends C++ with keywords and expressions specific to the vertex-vertex algebra. In order to be executed, a vv program is first translated to a C++ program, with the keywords and expressions specific to vv translated into calls to the vvlib library. This C++ program is then compiled into a dynamically linked library (DLL). The modeling program, called vvinterpreter, loads this DLL, runs, and produces the graphical output. This whole processing sequence is automated: from the user's perspective, the vvinterpreter treats the vv program as an input and runs accordingly. The methodology that we have used to implement the vv language closely follows that developed for L+C, an extension of C++ with programing constructs based on L-systems [13].

4 Subdivision algorithms

To illustrate the usefulness of the vertex-vertex algebra, we provide compact descriptions of several subdivision algorithms. These descriptions are expressed in the vv language and can be directly executed by vvinterpreter.

4.1 Insertion of a Vertex

One particularly simple routine that also happens to be of mush use in writing subdivision algorithms is the insertion of a new vertex between two neighbouring vertices. So, we first define a function that creates a new vertex x and inserts it between two given vertices p and q (Algorithm 1).

```
1 vertex insert(vertex p, vertex q) {
2   vertex x;
3   make {p, q} nb_of x;
4   replace p with x in q;
5   replace q with x in p;
6   return x;
7 }
```

Algorithm 1: Code and illustration of the insertion of a vertex x between vertices p and q. Vertex x replaces p as the neighbor of q and q as the neighbor of p; vertices p and q become neighbors of x.

4.2 Polyhedral Subdivision

One of the simplest possible subdivision schemes is polyhedral subdivision [28] for triangular meshes. The scheme simply inserts a new vertex at the midpoint of each edge such that each triangle in the mesh is subdivided into four co-planar triangles. While the geometry of the polygon mesh does not change, the topology is subdivided.

The program (Algorithm 2) that implements polyhedral subdivision consists of two loops. The first loop (lines 5 to 12) considers iterates over the existing pairs of neighbouring vertices in the set S and inserts a new vertex between them (Figure 5a). The new vertices are added to the set NV and are assigned a position at the midpoint of the pair of vertices.

The second loop (lines 13 to 18) inserts new edges by redefining the neighborhoods of the new points. The intervening neighborhoods and the result of insertion are shown in Figure 5b,c.

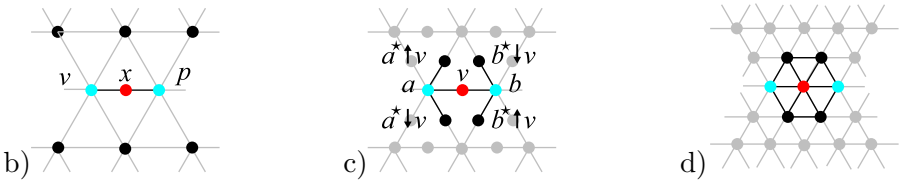


Figure 5: Illustration of the polyhedral subdivision algorithm implemented using vertex-vertex systems. a) The vv identification of points involved in the application of the mask to a new vertex x. b) The vv identification of vertices that will become neighbors of v. c) The mesh with all the new edges of v added.

```
 1 void polyhedral(mesh& S) {
 2   synchronize S;
 3   mesh NV;
 4
 5   forall v in S {
 6     forall p in ‘v {
 7       if (p < v) continue;
 8       vertex x = insert(v, p);
 9       x$pos = (p$pos + v$pos) / 2.0;
10       add x to NV;
11     }
12   }
13   forall v in NV {
14     vertex a = any in v;
15     vertex b = nextto a in v;
16     make {nextto v in b, b, prevto v in b,
17       nextto v in a, a, prevto v in a} nb_of v;
18   }
19   merge S with NV;
20 }
```

Algorithm 2: The polyhedral subdivision algorithm.

4.3 Loop algorithm

The Loop subdivision scheme [21] is topologically equivalent to the polyhedral subdivision scheme, in the sense that both operate on triangular meshes and subdivide a triangle into four triangles in an iteration step. The vertex-vertex implementations of both schemes have, therefore, a similar structure. The difference is in the placement of vertices. The Loop uses a mask to place new vertices and uses another mask to adjust the positions of old vertices (Figure 6). The implementation of the Loop subdivision algorithm is given by Algorithm 3.

4.4 Butterfly algorithm

The butterfly subdivision scheme for surfaces [7] is an interpolating scheme for triangular polygon meshes. The complete vv program that implements it for closed surfaces is given by Algorithm 4.

The algorithm for butterfly subdivision, like that for Loop subdivision,

```
 1 void loop(mesh& S) {
 2   double pi2 = 6.2832;
 3   synchronize S;
 4   mesh NV;
 5
 6   forall v in S {
 7     double n = valence v;
 8     double w = (0.625-pow(0.325 + 0.25*cos(pi2/n),2.0))/n;
 9     v$pos *= (1.0 - (double(n) * w));
10     forall p in ‘v {
11       v$pos += w * ‘p$pos;
12       if (p < v) continue;
13       vertex x = insert(v, p);
14       x$pos = 3.0/8.0 * ‘v$pos + 3.0/8.0 * ‘p$pos
15             + 1.0/8.0 * ‘(nextto p in ‘v)$pos
16             + 1.0/8.0 * ‘(prevto p in ‘v)$pos;
17       add x to NV;
18     }
19   }
20   forall v in NV {
21     vertex a = any in v;
22     vertex b = nextto a in v;
23     make {nextto v in b, b, prevto v in b,
24       nextto v in a, a, prevto v in a} nb_of v;
25   }
26   merge S with NV;
27 }
```

Algorithm 3: The Loop subdivision algorithm.

is topologicaly similar to the polyhedral subdivision. However, unlike Loop subdivision, the mask for the placement of new vertices requires the positions of vertices beyond the 1-ring. This mask and the corresponding vv identification of the intervening vertices are shown in Figure 7a,b.

4.5 $\sqrt{3}$ algorithm

Kobbelt's $\sqrt{3}$-subdivision [14] changes the topology of a triangular mesh in a manner different from the butterfly and Loop schemes (Figure 9). The corresponding vv implementation is given by Algorithm 5. In the first loop

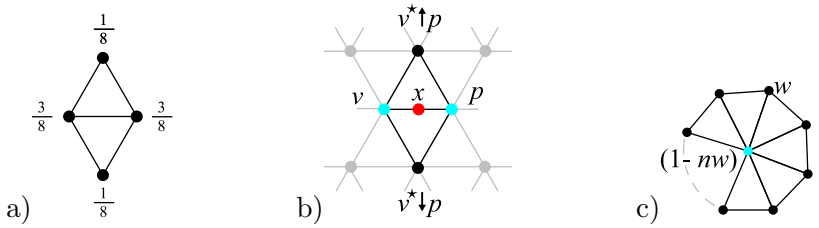


Figure 6: a) The Loop mask for a new vertex. b) The vv identification of points involved in the application of the mask to a new vertex x. c) The Loop mask for old vertices.

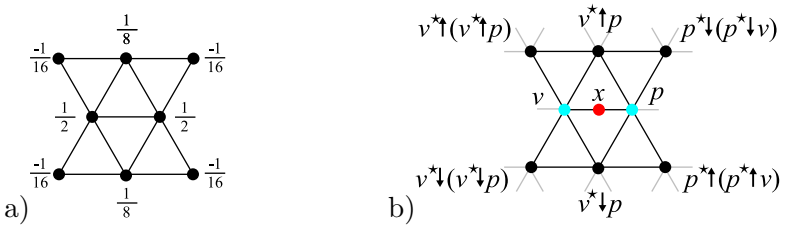


Figure 7: Illustration of the butterfly algorithm implemented using vertex-vertex systems. a) The mask. b) The vv identification of points involved in the application of the mask to a new vertex x.

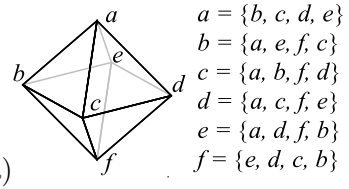
of the algorithm, a new vertex c is created at the centroid of each triangle (lines 11 to 15). The neighborhoods are then updated such that each triangle is divided into three, that is each vertex v, x, y of the original triangle is connected to c, and the vertices v, x, y form the neighborhood of c (lines 16 to 19, c.f. Figure 9b). In the second loop (lines 23 to 31, Figure 9c), the topology is updated by flipping all the edges between pairs of old vertices.

5 Conclusions

We have addressed the problem of specifying polygon mesh algorithms in a concise and intuitive manner. To this end, we introduced a set of operations for locally changing the topology of a mesh, and we defined these operations in terms of relations between mesh elements. We have focused on subdivision algorithms as an application area, and we have shown that the resulting vertex-vertex algebra leads to a very compact and intuitive specifications of some of the best known algorithms.

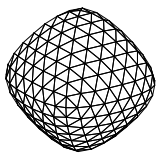
```
 1 void butterfly(mesh& S) {
 2   double k = 1.0/16.0, l = 1.0/8.0, m = 1.0/2.0;
 3   synchronize S;
 4   mesh NV;
 5
 6   forall v in S {
 7     forall p in ‘v {
 8       if (p < v) continue;
 9       vertex x = insert(v, p);
10       x$pos = m * ‘v$pos + m * ‘p$pos
11         + l * ‘(prevto p in ‘v)$pos
12         + l * ‘(nextto p in ‘v)$pos
13         - k * ‘(nextto (nextto p in ‘v) in ‘v)$pos
14         - k * ‘(nextto (nextto v in ‘p) in ‘p)$pos
15         - k * ‘(prevto (prevto p in ‘v) in ‘v)$pos
16         - k * ‘(prevto (prevto v in ‘p) in ‘p)$pos;
17       add x to NV;
18     }
19   }
20   forall v in NV {
21     vertex a = any in v;
22     vertex b = nextto a in v;
23     make {nextto v in b, b, prevto v in b,
24       nextto v in a, a, prevto v in a} nb_of v;
25   }
26   merge S with NV;
27 }
```

Algorithm 4: The butterfly subdivision algorithm.



$a = \{b, c, d, e\}$
$b = \{a, e, f, c\}$
$c = \{a, b, f, d\}$
$d = \{a, c, f, e\}$
$e = \{a, d, f, b\}$
$f = \{e, d, c, b\}$

a)



b)

Figure 8: The butterfly algorithm in action. (a) An initial polyhedron and the vertex-vertex specification of its topology. (b) The polyhedron after three subdivision steps.

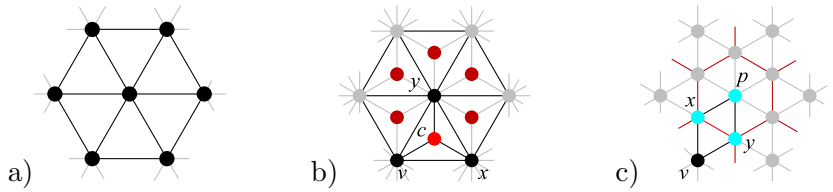


Figure 9: Mesh topology changes in the $\sqrt{3}$ scheme. a) A portion of the original mesh. b) The mesh after the insertion of central points, and subdivision of triangles. c) The mesh after the flip operation.

We have also designed vv, a programming language based on the vertex-vertex algebra, and we implemented a modeling environment in which vv programs can be executed. In addition to the subdivision algorithms described in this paper, we used vv to generate fractals and aperiodic tilings, simulate growth of multicellular biological structures, and create procedural textures on non-regular meshes. In these tests, we found vv programs extremely conducive to rapid prototyping and experimentation with polygon mesh algorithms.

Our implementation of the vertex-vertex algebra was guided by the elegance of programming constructs, rather than performance. For example, profiling of vv programs showed that approximately 50% of the algorithm execution time is spent on dynamic memory management. It is an interesting open question, if vertex-vertex systems could reach the speed of the fastest implementations of polygon mesh algorithms.

Another interesting class of problem is related to the temporal coordination of vertex-vertex operations. The synchronization mechanism introduced in Section 3.2.4 is in fact a method for simulating parallelism on a sequential machine. This suggests that it may be useful to extended vv with constructs for explicitly specifying parallel rather than sequential execution of operations. Such an extension could further clarify vv programs, and lead to their effective implementation on parallel processors with a suitable architecture.

References

[1] E. Akleman and J. Chen. Guaranteeing the 2-manifold property for meshes with doubly linked face list. *International Journal of Shape Modeling*, 5(2):149–177, 2000.

```
 1 void sqrt3(mesh& S) {
 2   synchronize S;
 3   mesh NV;
 4
 5   forall v in S {
 6     double pi2 = 6.28;
 7     double n = valence `v;
 8     double w = (4.0 - 2.0 * cos(pi2 / n)) / 9.0;
 9     v$pos *= (1.0 - w);
10     forall x in `v {
11       v$pos += `x$pos * w / n;
12       vertex y = nextto x in `v;
13       if (x < v || y < v) continue;
14       vertex c;
15       c$pos = (`v$pos + `x$pos + `y$pos) / 3.0;
16       make {v, x, y} nb_of c;
17       splice c after x in v;
18       splice c after y in x;
19       splice c after v in y;
20       add c to NV;
21     }
22   }
23   forall v in S {
24     forall p in `v {
25       if (p < v) continue;
26      vertex x = nextto p in v;
27       vertex y = prevto p in v;
28       splice y after v in x; splice x after p in y;
29       erase p from v; erase v from p;
30     }
31   }
32   merge S with NV;
33 }
```

Algorithm 5: The algorithm for $\sqrt{3}$ subdivision.

[2] E. Akleman, J. Chen, and V. Srinivasan. A new paradigm for changing topology during subdivision modeling. In *Pacific Graphics 2000*, pages 192–201, October 2000.

[3] E. Akleman, J. Chen, and V. Srinivasan. A prototype system for robust, interactive and user-friendly modeling of orientable 2-manifold meshes. In *Proceedings of Shape Modeling International 2002*, pages 43–50, May 2002.

[4] B. Baumgart. Winged edge polyhedron representation. Technical Report STAN-CS-320, Stanford University, 1972.

[5] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, 1978.

[6] G. Chaikin. An algorithm of high speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.

[7] N. Dyn, D. Levin, and J. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, 1990.

[8] J. Edmonds. A combinatorial representation of polyhedral surfaces (abstract). *Notices of the American Mathematical Society*, 7:646, 1960.

[9] R. Egli and N. F. Stewart. A framework for system specification using chains on cell complexes. *Computer-Aided Design*, 32:447–459, 2000.

[10] J.-L. Giavitto and O. Michel. Declarative definition of group indexed data structures and approximations of their domains. In *Proceedings of the 3rd ACM SIGPLAN Conference on Principles and Practice of Declarative Programming PPDP-01*, 2001.

[11] J.-L. Giavitto and O. Michel. MGS: A programming language for the transformation of topological collections. Research Report 61-2001, CNRS - Université d'Evry Val d'Esonne, Evry, France, 2001.

[12] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[13] R. Karwowski. *Improving the process of plant modeling: the L+C modeling language*. PhD thesis, University of Calgary, August 2002.

[14] L. Kobbelt. $\sqrt{3}$-subdivision. In *Computer Graphics*, 2000.

[15] W. Lickorish. Simplicial moves on complexes and manifolds. In *Proceedings of the Kirbyfest*, volume 2, pages 299–320, 1999.

[16] P. Lienhardt. Subdivisions de surfaces et cartes généralisées de dimension 2. *Informatique Théorique et Applications*, 25(2):171–202, 1991.

[17] P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-aided Design*, 23(1):59–82, 1991.

[18] P. Lienhardt. Subdivision par opérations locales, 2001. Manuscript, Université de Poitiers, November 2001.

[19] A. Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[20] A. Lindenmayer and G. Rozenberg. Parallel generation of maps: Developmental systems for cell layers. In V. Claus, H. Ehrig, and G. Rozenberg, editors, *Graph grammars and their application to computer science; First International Workshop*, Lecture Notes in Computer Science 73, pages 301–316. Springer-Verlag, Berlin, 1979.

[21] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, The University of Utah, August 1987.

[22] M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Maryland, 1988.

[23] R. Palmer and V. Shapiro. Chain models of physical behavior for engineering analysis and design. *Research in Engineering Design*, 5:161–184, 1993.

[24] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[25] P. Prusinkiewicz, F. Samavati, C. Smith, and R. Karwowski. L-system description of subdivision curves. Submitted, June 2002.

[26] G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation*. World Scientific, Singapore, 1997.

[27] M. Sabin. Subdivision surfaces, 2002. Shape Modeling International 2002 Tutorial Notes, 25 pp.

[28] E. Stollnitz, T. DeRose, and D. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufman Publishers, Inc., 1996.

[29] L. Velho. Stellar subdivision grammars. Submitted, January 2003.

[30] A. White. *Graphs, groups and surfaces.* North-Holland, Amsterdam, 1973.

[31] D. Zorin, P. Schröder, A. DeRose, L. Kobbelt, A. Levin, and W. Sweldens. Subdivision for modeling and animation, 2000. SIGGRAPH 2000 Course Notes 23.