Local specification of surface subdivision algorithms

Colin Smith, Przemyslaw Prusinkiewicz, and Faramarz Samavati
Department of Computer Science, University of Calgary
Calgary, Alberta, Canada T2N 1N4
{smithco|pwp|samavati}@cpsc.ucalgary.ca

Abstract

Many polygon mesh algorithms operate in a local manner, yet are formally specified using global indexing schemes. This obscures the essence of these algorithms and makes their specification unnecessarily complex, especially if the mesh topology is modified dynamically. We address these problems by defining a set of local operations on polygon meshes represented by graph rotation systems. We also introduce the vv programming language, which makes it possible to express these operations in a machine−readable form. The usefulness of the vv language is illustrated by the application examples, in which we concentrate on subdivision algorithms for the geometric modeling of surfaces. The algorithms are specified as short, intuitive vv programs, directly executable by the corresponding modeling software.

Reference

C. Smith, P. Prusinkiewicz, F. Samavati: Relational specification of subdivision algorithms. *Applications of Graph Transformations with Industrial Relevance* (AGTIVE 2003): *Lecture Notes in Computer Science* 3062, pp. 313−327

Local Specification of Surface Subdivision Algorithms

Colin Smith, Przemyslaw Prusinkiewicz and Faramarz Samavati

University of Calgary,
Calgary, Alberta, Canada T2N 1N4
{smithco,pwp,samavati}@cpsc.ucalgary.ca

Abstract. Many polygon mesh algorithms operate in a local manner, yet are formally specified using global indexing schemes. This obscures the essence of these algorithms and makes their specification unnecessarily complex, especially if the mesh topology is modified dynamically. We address these problems by defining a set of local operations on polygon meshes represented by graph rotation systems. We also introduce the vv programming language, which makes it possible to express these operations in a machine-readable form. The usefulness of the vv language is illustrated by the application examples, in which we concentrate on subdivision algorithms for the geometric modeling of surfaces. The algorithms are specified as short, intuitive vv programs, directly executable by the corresponding modeling software.

1 Introduction

Locality is one of the most fundamental characteristics of structured dynamical systems. It means that a neighborhood relation is defined on the elements of the system, and that each element changes its state according to its own state and the state of its neighbors. The elements positioned farther away are not considered. A challenging problem is the characterization and modeling of dynamical systems with a dynamical structure [1], in which not only the state of the elements of the system, but also their number and configuration, change over time. In this paper, we consider a class of such systems pertinent to geometric modeling and represented by surface subdivision algorithms.

Subdivsion algorithms generate smooth (at the limit) surfaces by iteratively subdividing polygon meshes. This process involves the creation of new vertices, edges, and faces. The operations on the individual mesh elements are described in local terms using *masks* [2], also referred to as *stencils* [3]. A mask is a graph that depicts a vertex of interest (either a newly created point or an old vertex being repositioned) and the neighbors that affect it. The new vertex position is determined as an affine combination[1] of the vertices identified by the mask.

Despite the locality and simplicity of the masks, formal descriptions of subdivision algorithms often rely on a global enumeration (indexing) of the polygon mesh elements [2]. Indices have the advantage of being the standard mathematical notation, conducive to stating and proving properties of subdivision algorithms. They are also

[1] An affine combination of n points $P_1, P_2, \ldots, P_n$ is an expression of the form $\alpha_1 P_1 + \alpha_2 P_2 + \cdots + \alpha_n P_n$, where the scalar coefficients α_i add up to one: $\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$. The meaning of the affine combination is derived from its transformation to the form $P = P_1 + \alpha_2(P_2 - P_1) + \cdots + \alpha_n(P_n - P_1)$, which is a well-defined expression of vector algebra [4, 5].

closely related to the array data structures supported by most programming languages. On the other hand, indices provide only an indirect access to the neighbors of an element in a polygon mesh: all mesh elements must be first enumerated and then index arithmetic must be performed to select a specific neighbor. This arithmetic becomes cumbersome for irregular meshes. Moreover, for dynamically changing mesh structures, indices may have to be updated after each iteration of the algorithm. The index notation is also too powerful: providing a unique index to each vertex makes it possible to access vertices at random, thus violating the locality constraint. We seek a notation that would be as intuitive as masks, but also sufficiently precise to formally specify and execute subdivision algorithms.

Within the confines of linear and branching structures, an example of such a notation is provided by L-systems. In addition to the generation of fractals and the modeling of plants [6], L-systems have recently been applied to the geometric modeling of subdivision curves [7]. Unfortunately, known extensions of L-systems to polygonal structures, namely *map L-systems* [6, 8] and *cell systems* [9], lack features needed for the specification of subdivision algorithms for surfaces. They do not offer a flexible control over geometry, and do not provide a mechanism for accessing context information. Limited geometric interpretations and a focus on the context-free case are also prevalent in the broader scope of graph grammars (c.f. [10, 11]).

Our proposed solution preserves the purely local operation of L-systems and graph grammars, but departs from their declarative character. Instead, context information is accessed and modeled structures are modified by sequences of imperative operations.

The ease of performing local operations on polygon meshes depends on their representation. Known examples of such representations, conducive to both local information gathering and mesh transformations, include the *winged-edge* [12], and *quad-edge* [13] representations. Pursuing objectives closer to ours, Egli and Stewart [14] applied *cellular complexes* [15] to specify Catmull-Clark [16] subdivision in an index-free manner, and Lienhardt [17] showed that local operations involved in subdivision algorithms can be defined using *G-maps* [18, 19]. More recently, Velho [20] proposed a method for describing subdivision algorithms using stellar operators [21] that act on a *half-edge* structure [22].

We have chosen yet another representation, based on the mathematical notion of *graph rotation systems* [23, 24]. A graph 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 [24]. Graph rotation systems have been introduced to computer graphics by Akleman, Chen and Srinivasan [25–27] 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.

We introduce *vertex-vertex systems*, related to the *adjacency-list* graph representation [28], as an alternative data structure based on the graph rotation systems. We also define the *vertex-vertex algebra* for describing local operations on the vertex-vertex systems, and VV, an extension of the C++ programming language, for expressing these operations in a machine-readable form. This leads us to the *language + engine* modeling

paradigm, which simplifies the implementation of individual algorithms by treating them as input to a multi-purpose modeling program. We illustrate the usefulness of this paradigm by presenting concise VV specifications of several subdivision algorithms.

2 Vertex-vertex systems

2.1 Definitions

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 3). 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*[2] 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 lie 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 and 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 rotation system.

A *polygon mesh* is a collection 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 [24].

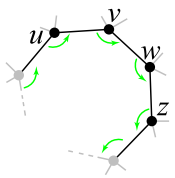

Fig. 1. A polygon identification in a graph rotation system.

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* [23, 24, 26], which is defined for connected graph rotation systems. It defines polygons of the mesh using the following algorithm (Fig. 1). 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

[2] Our terminology is motivated by the practice of referring to adjacent cells in a grid as neighbors.

that the resulting mesh is a uniquely defined, orientable, closed 2-manifold (see [24] for a proof).

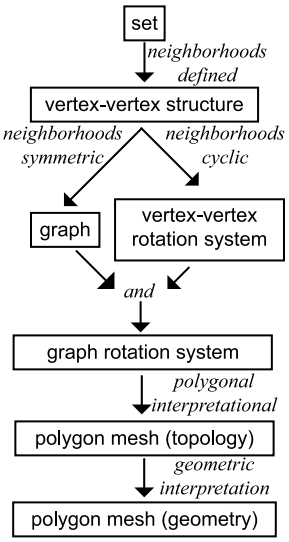

Fig. 2. Relations between notions pertinent to vertex-vertex systems

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 Fig. 2. It suggests that polygon meshes can be manipulated using three types of operations: set-theoretic, topological, and geometric operations. The most difficult problem is the manipulation of topology. We address it by introducing a set of operations 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 $v \notin u^\star$).

2.2 The vertex-vertex algebra

The *vertex-vertex algebra* consists of 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 2.3.

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

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 locally modify a vertex-vertex system. Definitions of these operations are given in Table 2. The last column points to the illustrations below the table.

Name	Math. notation	vv statement	Description	Note	Fig
Query operations					
membership	$x \in v^\star$	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	$\lvert v^\star \rvert$	valence v	returns the number of neighbors of vertex v		
Selection operations					
any	let $x \in v^\star$	any in v	returns a 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 a 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 $x \in v^\star$	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 after a in the neighborhood of v	5	f
splice before	$v^\star + x \prec a$	splice x before a in v	insert x 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 is listed twice.
4) No effect if $x \notin v^\star$. 5) No effect if $a \notin v^\star$; not defined (error reported) if $x = v$ or $x \in v^\star$.

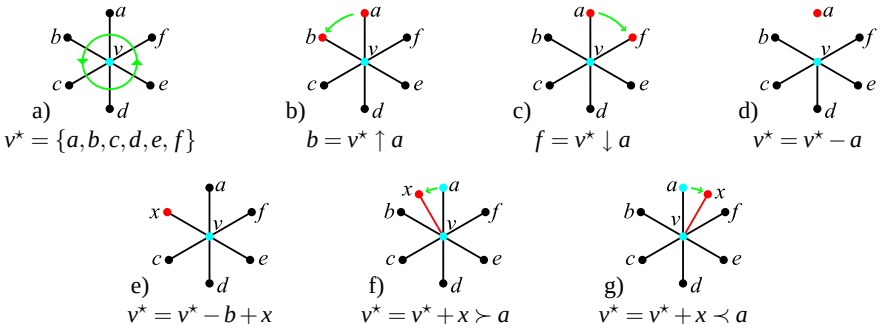

Table 2. Top: definition of the topological operations of the vertex-vertex algebra. Bottom: graphical interpretation of the selection and editing operations. a) Setting the initial neighborhood of vertex v. b-g) The results of various operations applied to v.

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 $\bar{v}$ 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.

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 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.

2.3 Implementation of vertex-vertex systems

We have implemented vertex-vertex systems as 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 implementing the vertex-vertex algebra. They are listed under the column 'vv statement' in Tables 1 and 2. All of the examples presented in this paper are actual code written in the vv language. To enhance code legibility, we set variable names in italics.

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. This approach is based on that introduced by Karwowski and Prusinkiewicz to translate and execute L-system-based programs in [29].

3 Subdivision algorithms

To illustrate the usefulness of the vertex-vertex algebra, we now provide compact descriptions of several subdivision algorithms. These descriptions can be directly executed by vvinterpreter.

3.1 Insertion of a Vertex

A simple routine that is of much use in writing subdivision algorithms is a function that creates a new vertex x and inserts it between two given vertices p and q (Fig. 3).

```
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  }
```

Fig. 3. The vv 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.

3.2 Polyhedral subdivision

One of the simplest subdivision algorithms is the polyhedral subdivision of triangular meshes [30]. The algorithm inserts a new vertex at the midpoint of each edge, and divides each triangle of the mesh into four co-planar triangles. While the overall shape of the initial polyhedron does not change, the faces are subdivided.

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

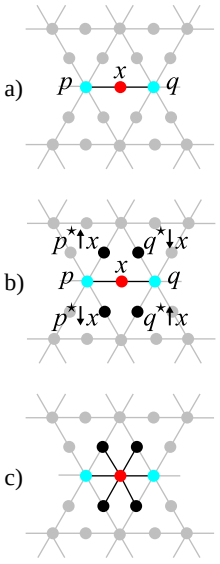

Fig. 4. Left: the polyhedral subdivision algorithm specified using vertex-vertex systems. Right: the vv identification of points involved in the creation of a new vertex x (a), the vv identification of vertices that will become neighbors of x (b), and the updated neighborhood of the new vertex x (c).

The vv program that implements one step of the polyhedral subdivision consists of two loops (Fig. 4). The first loop (lines 5 to 12) iterates over pairs of neighboring

vertices in the old vertex set S. The condition $p < q$ in line 7 assures that each vertex pair (i.e. edge of the polygon mesh) will be considered only once. New vertices are inserted at the midpoint of each edge (line 9) and added to the set NV (line 10). 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 on the right side of Figure 4. An example of a polygon mesh and the results of its polyhedral subdivision are shown in Figure 5.

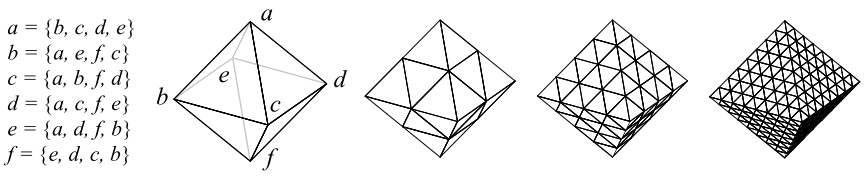

Fig. 5. From left to right: The VV specification of an initial polygon mesh topology, a sample polyhedron with that topology, and three steps of its polyhedral subdivision (with hidden lines eliminated).

3.3 Loop algorithm

The Loop subdivision scheme [31] is topologically equivalent to the polyhedral subdivision scheme, in the sense that both operate on triangular meshes and subdivide a triangular face into four triangles in every iteration step. The vertex-vertex implementations of both schemes have, therefore, a similar structure. The difference is in the positioning of vertices: the Loop case aims at constructing a smooth surface with a general shape controlled by the initial polyhedron (Fig. 6). To this end, the Loop algorithm places new vertices using a mask involving four old vertices, and repositions old vertices using another mask that incorporates all of their immediate neighbors. A VV implementation of the Loop subdivision algorithm for closed surfaces and the corresponding masks are given in Figure 7. The derivation of the coefficients of the masks is presented in [31].

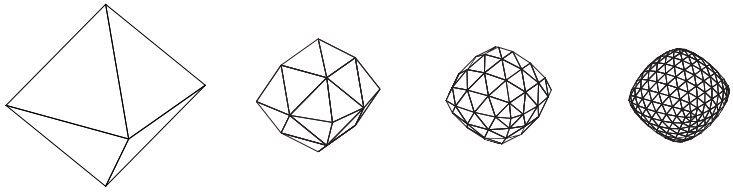

Fig. 6. An initial polygon mesh and the results of three iterations of Loop subdivision

```
 1  void loop(mesh& S) {
 2    double pi2 = 6.2832;
 3    synchronize S;
 4    mesh NV;
 5
 6    forall p in S {
 7      double n = valence p;
 8      double w = (5.0/8.0)/n
 9        −pow(3.0/8.0 + 1.0/4.0∗cos(pi2/n),2.0)/n;
10      p$pos ∗= 1.0 − (n ∗ w);
11      forall q in ‘p {
12        p$pos += w ∗ ‘q$pos;
13        if (p < q) continue;
14        vertex x = insert(p, q);
15        x$pos = 3.0/8.0 ∗ ‘p$pos + 3.0/8.0 ∗ ‘q$pos
16          + 1.0/8.0 ∗ ‘(nextto q in ‘p)$pos
17          + 1.0/8.0 ∗ ‘(prevto q in ‘p)$pos;
18        add x to NV;
19      }
20    }
21    forall x in NV {
22      vertex p = any in x;
23      vertex q = nextto p in v;
24      make {nextto x in q, q, prevto x in q,
25        nextto x in p, p, prevto x in p} nb_of x;
26    }
27    merge S with NV;
28  }
```

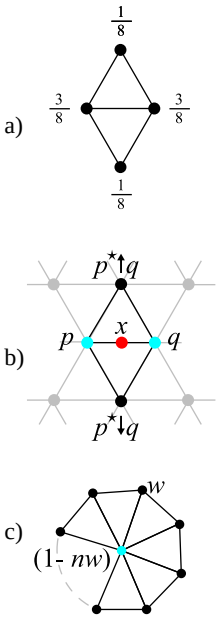

Fig. 7. Left: the vv implementation of the Loop subdivision algorithm. Right: illustration of the algorithm. a) The Loop mask for a new vertex. Vertex labels are the weights used in the affine combinations of vertex positions. b) The vv identification of vertices involved in the application of the mask to a new vertex x. c) The Loop mask for old vertices.

3.4 Butterfly algorithm

The butterfly subdivision algorithm [32], like that for Loop subdivision, is topologically equivalent to the polyhedral subdivision. In contrast to the Loop subdivision, however, which approximates the shape of the initial polyhedron, the butterfly algorithm is an interpolating scheme. Consequently, the old vertex positions are not adjusted in the course of the algorithm. In order to produce a smooth limit surface, the butterfly algorithm uses a more extensive mask for the new vertices, which includes points outside the immediate neighborhood of the subdivided edge. This mask and the complete vv implementation of the butterfly algorithm for closed surfaces are presented in Figure 8. An example application of the algorithm is illustrated in Figure 9.

3.5 $\sqrt{3}$ algorithm

Kobbelt's $\sqrt{3}$-subdivision algorithm [33] is an example of a scheme that changes the topology of a triangular mesh in a manner different from the polyhedral subdivision. The vv specification of the $\sqrt{3}$-subdivision algorithm is given by Figure 10. In the first

```
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 p in S {
7      forall q in ‘p {
8        if (p < q) continue;
9        vertex x = insert(p, q);
10       x$pos = m ∗ ‘p$pos + m ∗ ‘q$pos
11         + l ∗ ‘(prevto q in ‘p)$pos
12         + l ∗ ‘(nextto q in ‘p)$pos
13         − k ∗ ‘(nextto (nextto q in ‘p) in ‘p)$pos
14         − k ∗ ‘(nextto (nextto p in ‘q) in ‘q)$pos
15         − k ∗ ‘(prevto (prevto q in ‘p) in ‘p)$pos
16         − k ∗ ‘(prevto (prevto p in ‘q) in ‘q)$pos;
17       add x to NV;
18     }
19   }
20   forall x in NV {
21     vertex p = any in x;
22     vertex q = nextto p in x;
23     make {nextto x in q, q, prevto x in q,
24       nextto x in p, p, prevto x in p} nb_of x;
25   }
26   merge S with NV;
27 }
```

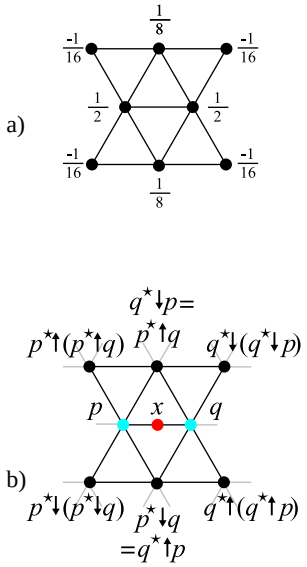

Fig. 8. Left: the VV implementation of the butterfly algorithm. Right: the mask (a) and VV identification (b) of points involved in its application to a new vertex x.

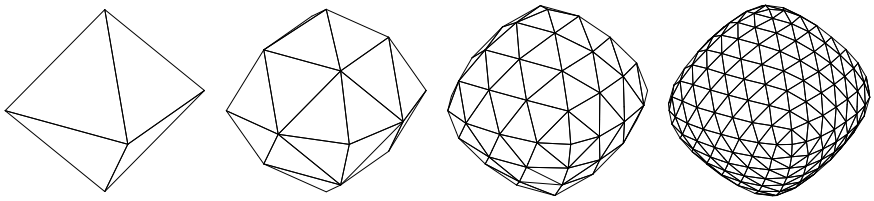

Fig. 9. An initial polygon mesh and the results of three iterations of butterfly subdivision

loop (lines 11 to 15), a new vertex c is created at the centroid of each triangle. 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). In the second loop (lines 23 to 31), the topology is updated by “flipping” all the edges between pairs of old vertices. An example of the operation of the algorithm is shown in Figure 11.

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

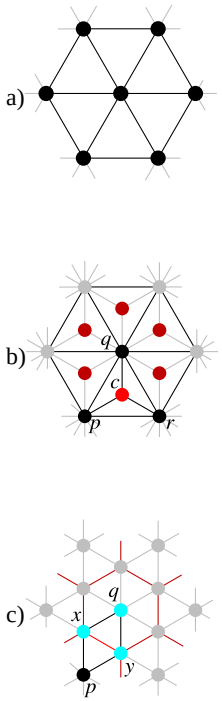

Fig. 10. Left: the algorithm for $\sqrt{3}$-subdivision. Right: a portion of the original mesh (a) after the insertion of central points and subdivision of triangles (b) and after the flip operation (c).

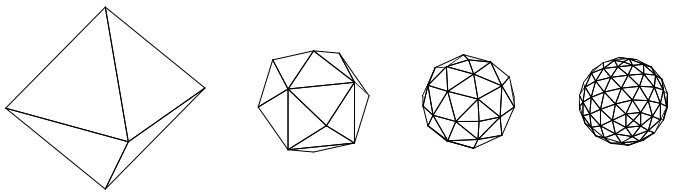

Fig. 11. Example of subdivision using the $\sqrt{3}$ algorithm

4 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 local terms. We have focused on subdivision algorithms as an application area, and we have shown that the resulting vertex-vertex algebra leads to very compact and intuitive specifications of some of the best known algorithms.

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 whether vertex-vertex systems could reach the speed of the fastest implementations of polygon mesh algorithms.

Another interesting class of problems is related to the temporal coordination of vertex-vertex operations. The synchronization mechanism introduced in Section 2.2 is in fact a method for simulating parallelism on a sequential machine. This suggests that it may be useful to extend 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. Finally, the problem of providing a declarative, grammar-like method for specifying subdivision algorithms remains open. Such specification, if possible, may provide the ultimately concise and clear specification of these algorithms.

References

1. Giavitto, J.L., Michel, O.: MGS: A programming language for the transformation of topological collections. Research Report 61-2001, CNRS - Université d'Evry Val d'Esonne (2001)
2. Zorin, D., Schröder, P., DeRose, T., Kobbelt, L., Levin, A., Sweldens, W.: Subdivision for modeling and animation. In: SIGGRAPH Course Notes, New York, ACM (2000)
3. Sabin, M.: Subdivision surfaces. Shape Modeling International tutorial notes (2002) 25pp.
4. DeRose, T.: A coordinate-free approach to geomeric programming. In Strasser, W., Seidel, H.P., eds.: Theory and practice of geometric modeling. Springer, Berlin (1989) 291–305
5. Goldman, R.: On the algebraic and geometric foundations of computer graphics. ACM Transactions on Graphics **21** (2002) 52–86
6. Prusinkiewicz, P., Lindenmayer, A.: The algorithmic beauty of plants. Springer-Verlag, New York (1990)
7. Prusinkiewicz, P., Samavati, F., Smith, C., Karwowski, R.: L-system description of subdivision curves. International Journal on Shape Modeling **9** (2003) 41–59
8. Lindenmayer, A., Rozenberg, G.: Parallel generation of maps: Developmental systems for cell layers. In Claus, V., Ehrig, H., Rozenberg, G., eds.: Graph grammars and their application to computer science; First International Workshop. Lecture Notes in Computer Science 73. Springer, Berlin (1978) 301–316
9. de Boer, M., Fracchia, F., Prusinkiewicz, P.: A model for cellular development in morphogenetic fields. In Rozenberg, G., Salomaa, A., eds.: Lindenmayer systems: Impacts

on theoretical computer science, computer graphics, and developmental biology. Springer, Berlin (1992) 351–370
10. Rozenberg, G., ed.: Handbook of graph grammars and computing by graph transformation. World Scientific, Singapore (1997)
11. Ehrig, H., Engles, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools. World Scientific, Singapore (1999)
12. Baumgart, B.: Winged-edge polyhedron representation. Technical Report STAN-CS-320, Stanford University (1972)
13. Guibas, L., Stolfi, J.: Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. ACM Transactions on Graphics **4** (1985) 74–123
14. Egli, R., Stewart, N.F.: A framework for system specification using chains on cell complexes. Computer-Aided Design **32** (2000) 447–459
15. Palmer, R., Shapiro, V.: Chain models of physical behavior for engineering analysis and design. Research in Engineering Design **5** (1993) 161–184
16. Catmull, E., Clark, J.: Recursively generated B-spline surfaces on arbitrary topological meshes. Computer Aided Design **10** (1978) 350–355
17. Lienhardt, P.: Subdivision par opérations locales. Manuscript, Université de Poitiers (2001)
18. Lienhardt, P.: Subdivisions de surfaces et cartes généralisées de dimension 2. Informatique Théorique et Applications **25** (1991) 171–202
19. Lienhardt, P.: Topological models for boundary representation: a comparison with *n*-dimensional generalized maps. Computer-Aided Design **23** (1991) 59–82
20. Velho, L.: Stellar subdivision grammars. In: Proceedings of Eurographics Symposium on Geometry Processing, Eurographics Association (2003) 12pp.
21. Lickorish, W.B.R.: Simplicial moves on complexes and manifolds. Geometry and Topology Monographs **2** (1999) 299–320
22. Mäntylä, M.: An introduction to solid modeling. Computer Science Press, Rockville (1988)
23. Edmonds, J.: A combinatorial representation of polyhedral surfaces (abstract). Notices of the American Mathematical Society **7** (1960) 646
24. White, A.: Graphs, groups and surfaces. North-Holland, Amsterdam (1973)
25. Akleman, E., Chen, J.: Guaranteeing the 2-manifold property for meshes with doubly linked face list. International Journal of Shape Modeling **5** (2000) 149–177
26. Akleman, E., Chen, J., Srinivasan, V.: A new paradigm for changing topology during subdivision modeling. In: Proceedings of Pacific Graphics. (2000) 192–201
27. Akleman, E., Chen, J., Srinivasan, V.: A prototype system for robust, interactive and user-friendly modeling of orientable 2-manifold meshes. In: Shape Modeling and Applications – Proceedings of Shape Modeling International. (2002) 43–50
28. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to algorithms. Second Edition. MIT Press, Cambridge, MA (2001)
29. Karwowski, R., Prusinkiewicz, P.: Design and implementation of the L+C modeling language. Electronic Notes in Theoretical Computer Science **86.2** (2003) 19pp.
30. Stollnitz, E., DeRose, T., Salesin, D.: Wavelets for computer graphics. Morgan Kaufman, San Francisco (1996)
31. Loop, C.: Smooth subdivision surfaces based on triangles. Master's thesis, The University of Utah (1987)
32. Dyn, N., Levin, D., Gregory, J.: A butterfly subdivision scheme for surface interpolation with tension control. ACM Transactions on Graphics **9** (1990) 160–169
33. Kobbelt, L.: $\sqrt{3}$-subdivision. In: Proceedings of SIGGRAPH, ACM (2000) 103–112