Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

A vertex-centric representation for adaptive diamond-kite meshes

Luiz Henrique de Figueiredo

IMPA, Rio de Janeiro, Brazil

ARTICLE INFO

Article history:
Received February 5, 2024
Revised March 14, 2024
Final version March 20, 2024

ABSTRACT

We describe a concise representation for adaptive diamond-kite meshes based solely on the vertices and their stars. The representation is exact because it uses only integers, is much smaller than standard topological data structures, and is highly compressible. All topological elements are reconstructed in expected constant time per element.

Keywords: mesh representation, geometric modeling, data structures

1. Introduction

Quadrilateral meshes have advantages over triangle meshes in several applications, including modeling, texturing, and physically-based simulations, where their numerical properties lead to more accurate and faster results. Moreover, high-quality quadrilateral meshes produce even better results [1].

Eppstein [5] described *adaptive diamond-kite meshes*, a family of planar quadrilateral meshes that can be refined recursively using local subdivision operations and are based solely on two kinds of quadrilaterals: *diamonds*, rhombi with 60° and 120° angles, and *kites* with 60°, 90°, and 120° angles. Adaptive diamond-kite meshes have several properties that are useful in applications, such as using faces of bounded aspect ratio and being invariant under Laplacian smoothing [5]. Fig. 1 shows an adaptive diamond-kite mesh refined around an implicit curve.

The rigid geometry and topology of adaptive diamond-kite meshes suggest that they should admit computational representations that are more concise than explicitly listing all vertices, edges, and faces of the mesh and their adjacency relationships, as in standard topological data structures [3]. As far as we know, there are no specialized representations for adaptive diamond-kite meshes in the literature, despite their potential in applications. In fact, specialized representations for quadrilateral meshes are rare: previous work on mesh representations either targets general meshes or specializes to triangle meshes [3]. There has been some work on compression of quadrilateral meshes [10, 4], on compact representations of general embedded planar graphs [11, 8], and on representations based on vertex stars for planar meshes [9] and for general cell complexes [7].

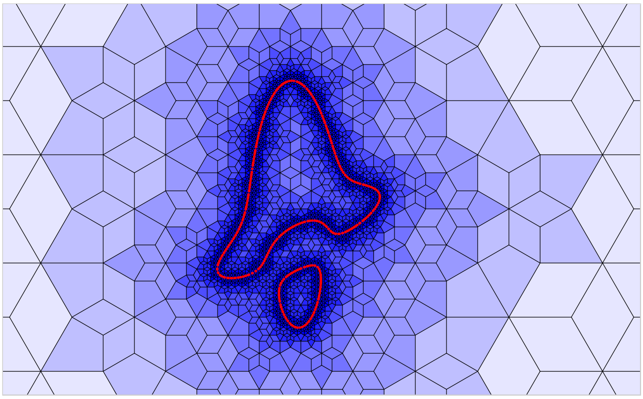


Fig. 1. An adaptive diamond-kite mesh refined around an implicit curve. The faces are colored according to their refinement depth.

In this paper, we describe in detail a concise representation for adaptive diamond-kite meshes based solely on the vertices and their stars. It is inspired by a recent vertex-centric representation for periodic tilings of the plane by regular polygons [12]. Our representation is: *exact*, because it uses integers for both the geometry and the topology; *concise*, much smaller than standard topological data structures; and *geometrically meaningful* in that it replaces explicit adjacency relations with a concise description of vertex stars. Storing the vertices in a hash table allows the reconstruction of the edges and faces adjacent to a vertex in expected constant time and so the reconstruction of the mesh in expected linear time. Our code is publicly available [2].

2. Adaptive diamond-kite meshes

We recall now the main concepts of adaptive diamond-kite meshes: their geometry and topology and the local subdivision operation. For further details, see the paper by Eppstein [5]. The main goal here is to lay down the basis of our representation: a complete geometric description of the stars of the internal vertices of the mesh. We shall discuss boundary vertices later.

Base mesh. An adaptive diamond-kite mesh starts with a *base mesh*: a finite rhombille tiling formed by subdividing a hexagonal tiling using three diamonds per hexagon (see Fig. 2). The base mesh has only diamonds; kites appear during refinement. In applications, a suitable base mesh is obtained by applying a similarity transformation to the *standard base mesh*: the one having edges of unit length and horizontal hexagons, as in Fig. 2. For concreteness, and without loss of generality, we focus here on that standard base mesh.

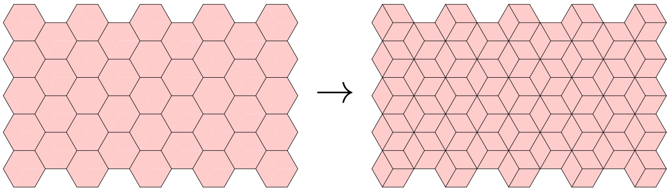


Fig. 2. The base mesh (right) is a finite rhombille tiling formed by subdividing a hexagonal tiling (left) using three diamonds per hexagon.

Faces. By definition, diamond-kite meshes have only two kinds of faces: *diamonds*, rhombi with 60° and 120° angles, and *kites*, with 60°, 90°, and 120° angles (see Fig. 3). A diamond has all sides of the same length. A kite has sides of length L and ρL, where $\rho = \tan(30°) = \frac{1}{\sqrt{3}} \approx 0.577$.

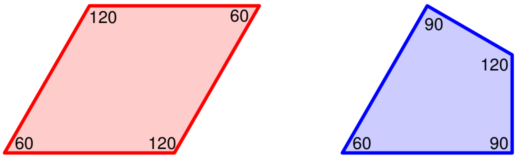


Fig. 3. The faces in diamond-kite meshes: diamonds (left) and kites (right).

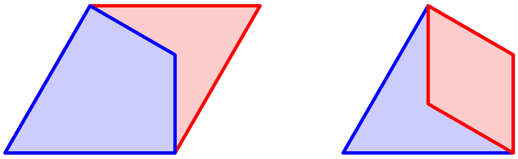


Fig. 4. A kite fits inside a diamond (left); a diamond fits inside a kite (right).

Diamonds and kites fit well together. As shown in Fig. 4, a diamond contains a kite: divide the diamond along its shortest diagonal into two equilateral triangles and take the barycenter of one triangle as the 120°-angle vertex of the kite. Similarly, a kite contains a smaller diamond: divide the kite along its shortest diagonal into two triangles and take the barycenter of the equilateral triangle as the other 120°-angle vertex of the diamond. These constructions support the recursive refinement of adaptive diamond-kite meshes, as we shall see presently.

Degrees. Since the smallest face angle is 60°, the maximum vertex degree in a diamond-kite mesh is 6. Since the largest face angle is 120°, the minimum vertex degree is 3. Adaptive diamond-kite meshes have vertices of all degrees between 3 and 6. The base mesh has only vertices of degree 3 and 6; vertices of degree 4 and 5 appear during refinement. (Recall that we focus here on internal vertices.) We exploit this narrow range of degrees in our representation.

Refinement. Adaptive meshes evolve by recursively applying topological refinement procedures to elements selected by application-specific refinement criteria. In adaptive diamond-kite meshes, refinement is applied to selected vertices of degree 6 using the local subdivision operation detailed below. Eppstein [5] also proposed a prerequisite structure of replacement operations that ensures smooth transitions during refinement so that adjacent faces differ by at most one level of refinement, as in restricted or balanced quadtrees [13] (see Fig. 1).

Subdivision. The main step for refining adaptive diamond-kite meshes is a *local subdivision operation* that subdivides and remeshes the faces around a selected *central vertex* of degree 6, as illustrated in Fig. 5. The mesh is unaffected outside these faces. Six *new vertices* (in green) are created at the barycenter of each equilateral triangle defined by the central vertex (in black) and two consecutive *adjacent vertices* (in red). The old adjacent vertices and the new vertices are then combined into six congruent diamonds (in red) around the central vertex. The new vertices become adjacent vertices and the old vertices become *opposite vertices*. From the perspective of the original opposite vertices (in blue), the original faces (in gray) are replaced by quadrilaterals (in blue) that switch types: diamonds become smaller kites and kites become smaller diamonds, as in Fig. 4. The mesh resulting after a local subdivision operation is thus still a diamond-kite mesh. The new edges around the central vertex are rotated 30° with respect to the original edges. These new central edges have length ρL, where ρ is the ratio between the sides of a kite and L is the length of the original central edges.

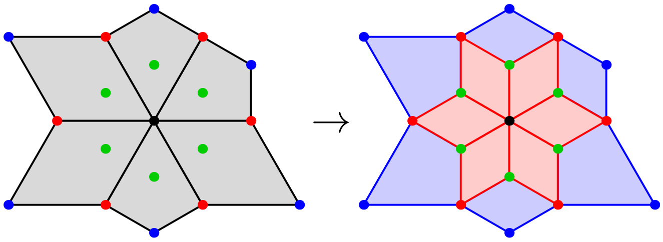


Fig. 5. The local subdivision operation in adaptive diamond-kite meshes. The six faces around a vertex of degree 6 (left) are subdivided and remeshed into twelve new faces (right). The new central faces are all diamonds. The original faces can be any combination of diamonds and kites.

Stars. In general meshes, the star of a vertex is the set of all topological elements adjacent to it. We adopt a simpler definition here: the *star* of a vertex is the circular sequence of vertices that are adjacent to it. Diamond-kite meshes have rigid geometry because they have only two kinds of faces. Rigid geometry implies rigid topology. In particular, the possible vertex stars are restricted to a small set. Indeed, the distribution of angles around a vertex is constrained by the integer solutions of the equation $60x + 90y + 120z = 360$, resulting in 11 possible arrangements of angles up to cyclic permutations (see Table 1). Moreover, the angles around a vertex determine its adjacent vertices up to scale: edges making angles of 60° or 120° have the same length; edges making angles of 90° have lengths whose ratio is ρ (see Fig. 3). These facts already suggest that diamond-kite meshes admit concise vertex-centric representations based on stars, the central idea in our representation. Adaptive diamond-kite meshes are even more severely constrained: the only possible arrangements of angles around a vertex are those shown in Fig. 6, exactly one for each degree. This is an important consequence of how the local subdivision operations works [5]. Thus, there is exactly one possible star for each degree, up to orientation and scale. This rigidity is the key for the conciseness of our representation.

x	y	z	degree	arrangements of angles		
0	0	3	3	ccc		
0	4	0	4	bbbb		
1	2	1	4	abbc	abcb	acbb
2	0	2	4	aacc	acac	
3	2	0	5	aaabb	aabab	
4	0	1	5	aaaac		
6	0	0	6	aaaaaa		

Table 1. The integer solutions of $60x + 90y + 120z = 360$ give 11 possible arrangements of angles around a vertex in a diamond-kite mesh. These arrangements are coded with a = 60°, b = 90°, c = 120°. Only ccc, abcb, aaabb, aaaaaa occur in adaptive diamond-kite meshes; see Fig. 6.

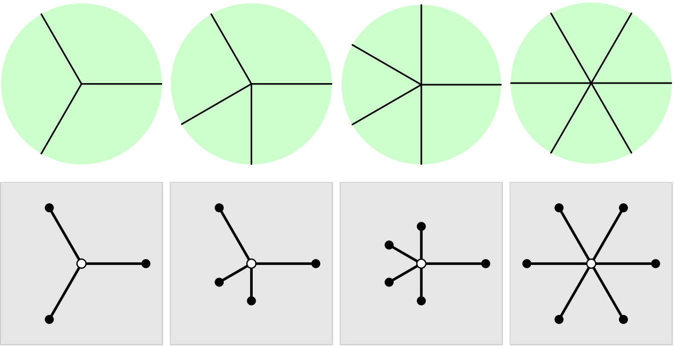


Fig. 6. Top: The possible arrangements of angles around a vertex in adaptive diamond-kite meshes. They are coded ccc, abcb, aaabb, aaaaaa in Table 1. Bottom: The only vertex stars for each degree, up to orientation and scale.

3. Background and inspiration

A representation for a polygonal mesh has two main components: geometry and topology. Because the faces are polygons, the edges are straight, and so all geometry is concentrated on the vertices, which are represented by suitable coordinates, typically Cartesian coordinates. The topology records selected adjacency relations between the vertices, edges, and faces of the mesh.

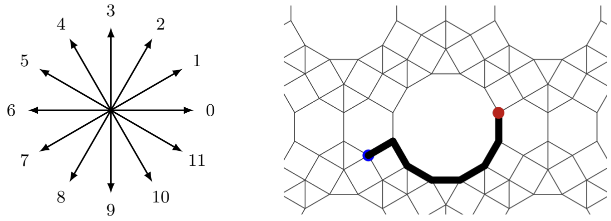


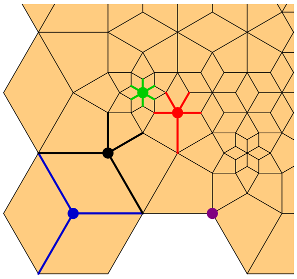
Fig. 7. Left: The twelve basic directions are given by ω^k for $k = 0, \ldots 11$. Right: Every vertex in a tiling of the plane by regular polygons can be reached by following a path from the origin along the edges of the tiling and so are given by integer polynomial expressions in ω (from [12]).

The art in representing meshes is choosing a subset of those adjacency relations such that all others are easily deduced from that subset [3]. In general meshes, a vertex can belong to several faces and a face can have several vertices. Therefore, standard representations for general meshes focus on the edges because they have exactly two adjacent vertices and at most two adjacent faces. Triangle and quadrilateral meshes are more rigid and admit representations based on faces and vertices. Tilings by regular polygons are even more rigid and admit representations based solely on vertices. Our representation for adaptive diamond-kite meshes is inspired by a recent concise vertex-centric representation for periodic tilings of the plane by regular polygons [12]. We briefly outline here the two features of that representation that are relevant to us: the choice of coordinates for the vertices and a data structure for storing them.

To describe the geometry of a tiling of the plane by regular polygons, one needs to give coordinates to its vertices. Complex numbers provide a rich and convenient framework for that task. First, scale and rotate the tiling so that all edges have unit length and are aligned with 12 *basic directions* given naturally by the 12th roots of unity in the complex plane (see Fig. 7). The basic directions are the powers of ω, where $\omega = \exp(\frac{2\pi i}{12}) = \frac{\sqrt{3}+i}{2}$ is the principal 12th root of unity. Next, translate the tiling so that the origin is at a vertex. Then, every vertex can be reached by following a path from the origin along the edges of the tiling (see Fig. 7). Since the edges have unit length and are aligned with the powers of ω, these paths are given by integer polynomial expressions in ω of degree less than 12. Therefore, every vertex has a representation in the ring of *cyclotomic integers* $\mathbf{Z}[\omega]$, the set of all integer polynomial expressions in ω. These expressions can be reduced to a unique canonical form of degree at most 3 because $\omega^4 = \omega^2 - 1$. Thus, every vertex can be written uniquely as $a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3$, with $a_i \in \mathbf{Z}$. We say that the vertex has *lattice coordinates* $[a_0, a_1, a_2, a_3]$. Finally, store the vertices of the tiling in a hash table, named the *cloud*, using their lattice coordinates as keys. The main feature of the cloud is that one can test whether a vertex exists at a given point in expected constant time. This allows the reconstruction of all vertices, edges, and faces around a given vertex in expected constant time [12].

For describing the topology of the tiling, the key observation is that the vertices adjacent to a given vertex are exactly its nearest neighbors, found at unit distance along the basic directions. By probing the cloud at these 12 locations, those vertices can be found in circular order in expected constant time, without explicit knowledge about the star of the vertex. More precisely, to find the vertices adjacent to a vertex v, check whether $v+\omega^k$ is in the cloud for $k = 0,\dots,11$. The lattice coordinates of $v+\omega^k$, needed for querying the cloud, are easily computed from those of v and ω^k. Thus, that representation for tilings stores no edges or faces nor their topological relations because they can all be readily deduced from the geometry of the vertices [12].

In summary, the representation for tilings [12] introduced lattice coordinates for vertices and the cloud for storing them. Both are key features for exact and efficient reconstruction of the tilings. In our representation of adaptive diamond-kite meshes, we retain the cloud for storing the vertices but we modify their lattice coordinates to reflect mesh refinement. We also provide each vertex with data to allow the reconstruction of its star.



v	a	b	m	d	k	n
🔵	0	0	0	3	0	0
⚫	0	1	0	4	6	0
🔴	2	5	1	5	9	1
🟢	0	2	0	6	1	3
🟣	2	0	0	0	0	0

Fig. 8. Our representation of some vertices and theirs stars. Their geometry is given by 3-adic lattice coordinates $[a,b,m]$ and their topology is given by their degree d, and the orientation k and level n of their stars.

4. Our representation

We exploit the rigidity of adaptive diamond-kite meshes to design a representation that is based solely on the vertices and their stars. The rigidity is the fact that there is exactly one possible star for each degree, up to orientation and scale (see Fig. 6). Thus, in our representation we store, for each vertex in the mesh, its modified lattice coordinates and the type, orientation, and scale of its star. This data directly represents the geometry of the mesh and indirectly represents its topology. Here are the details.

Base mesh. The vertices of the base mesh belong to an equilateral triangular mesh, a tiling of the plane by regular polygons, and so can be given lattice coordinates. Moreover, the edges of the base mesh are aligned with the 6 basic directions given by the powers of ω^2. Therefore, the coefficients of ω and ω^3 in the lattice coordinates are always zero and so the vertices can be represented in the simpler set $\mathbf{Z}[\omega^2] = \{a + b\omega^2 : a,b \in \mathbf{Z}\}$. The coordinates $[a,b]$ for a vertex $v = a + b\omega^2$ of the base mesh are still called its *lattice coordinates*. We now generalize those coordinates for the vertices created during refinement.

3*-adic lattice coordinates.* The only geometric operation used in refinement is the computation of barycenters of triangles to create new vertices in the local subdivision operation. The barycenter of a triangle uvw is given by $\frac{1}{3}(u+v+w)$. Therefore, every vertex in the mesh can be represented as $\frac{1}{3^m}(a+b\omega^2)$, where a,b,m are integers with $m \geq 0$. We say that the vertex has *rational* 3*-adic* lattice coordinates $[a,b,m]$; this is the geometric data attached to the vertex. The division by a power of 3 reflects the repeated computation of barycenters in the local subdivision operations. We call m the *depth* of the vertex: it is essentially the refinement depth at which the vertex is created and it does not change when the mesh is refined. The vertices of the base mesh have depth 0. To avoid ambiguities, especially when indexing the cloud, we use *normalized* 3-adic lattice coordinates by eliminating all common factors of 3 in a and b when $m > 0$, and adjusting m accordingly.

Topology. The vertices adjacent to a given vertex lie along the basic directions, but they are not necessarily its nearest neighbors, even when they are all at the same distance. For instance, in the standard base mesh every internal vertex has 6 nearest neighbors but not all internal vertices have degree 6, many have degree 3; also, the internal vertices of degree 3 fall into two classes where their stars differ by a rotation of 60°. Moreover, in refined meshes, the vertices adjacent to a vertex of degree 4 or 5 lie at different distances (see Fig. 6). Nevertheless, we can avoid listing adjacent vertices explicitly because there is exactly one possible star for each degree, up to orientation and scale.

Standard stars. For concreteness and convenience, we define *standard stars* for each degree in all 12 orientations, which we denote by $\operatorname{star}(d,k)$, where d is the degree and k is the orientation. Fig. 6 (bottom) defines $\operatorname{star}(d,0)$, the standard stars of each degree at orientation 0. Then $\operatorname{star}(d,k)$ is obtained by rotating $\operatorname{star}(d,0)$ by $30k$ degrees around its central vertex. The orientation of the star of each vertex in the mesh is determined with respect to the standard star of the same degree.

Stars. The topological data attached to each vertex is the type, orientation, and scale of its star. The *type* is determined by its degree d (see Fig. 6). The *orientation* is the integer k between 0 and 11 such that its star is completely aligned with $\operatorname{star}(d,k)$, except for scale. The *scale* is given by the *level* of the vertex. The level gives the length of the longest edge around the vertex: if the level is n, then the longest edge has length $L_n = \rho^n L_0$, where L_0 is the length of the edges in the base mesh. (By definition, $L_0 = 1$ in the standard base mesh.) All edges around a vertex of level n have length either L_n or $L_{n+1} = \rho L_n$ (see Fig. 6). The vertices of the base mesh are at level 0. The level of a vertex can increase when the mesh is refined.

Boundary vertices. The restricted stars shown in Fig. 6 only apply to internal vertices. The stars of boundary vertices have different restrictions that depend on the shape of the base mesh. To avoid this complication, we represent only the geometry of boundary vertices using their lattice coordinates, artificially giving them degree 0. We assume that every boundary vertex of the base mesh belongs to a face having an internal vertex. Then, the edges and faces adjacent to boundary vertices are implied by those of the internal vertices adjacent to them. The assumption on boundary vertices reflects a limitation of adaptive diamond-kite meshes: they need a base mesh with many internal vertices so that the refinement can advance. Except for that restriction, the base mesh may have multiple components and holes.

Abstract, internal, and external representations. Abstractly, our representation is a set of records $\langle a,b,m,d,k,n \rangle$, one for each vertex of the mesh, giving their geometry by 3-adic lattice coordinates $[a,b,m]$, and their topology by their degree d, and the orientation k and level n of their stars (see Fig. 8). Concretely, this abstract representation is supported by a suitable data structure, which we call an *internal representation*. We have found it very convenient to use the *cloud*, a hash table that stores vertices indexed by their normalized lattice coordinates [12]. We assume that the cloud answers queries in expected constant time; the details of hashing are otherwise unimportant. Our representation can easily be saved to a file as a list of records, in a suitable format. We call this an *external representation*. Standard CSV files are convenient text files that are easy to save, load, inspect, and share, but ad-hoc binary files can be used if reduced space or speed of loading are essential. External representations record the *complete* data required to rebuild an internal representation: no post-processing is needed. Standard mesh formats, such as OBJ and OFF, require extensive post-processing to rebuild a topological data structure. Below, we propose two variants for external representation: a *reduced* representation that takes about half the size of the full representation and a *normalized* representation that is useful for comparing models.

Reduced representation. Our representation is already compact: it represents only the vertices — no edges, no faces, no adjacency relations. Nevertheless, there is room for improvement because the edges are implicitly represented twice in vertex stars. Therefore, every internal vertex and its star can be reconstructed from its neighbors and their stars. Thus, each individual vertex is redundant and need not be represented, and so can be removed. However, we cannot remove too many vertices because the remaining vertices should enable the reconstruction of all redundant vertices and their stars. Finding the best set of redundant vertices to remove is probably a hard mesh decimation problem related to the minimum vertex cover problem in graphs, a classical optimization problem that is NP-hard. However, we do not need to find the best set of redundant vertices: the vertices of degree 3 are by far the most frequent vertices, comprising about half of all vertices (see Table 2, top). Indeed, since each local subdivision operation generates six new vertices of degree 3 adjacent to a vertex of degree 6 (Fig. 5), we expect that the vertices of degree 3 at the highest refinement depth to be the most numerous (see Table 2, bottom). Moreover, virtually all vertices of degree 3 are adjacent to a vertex of degree 6 and so are easily reconstruted (see the details in §5). In conclusion, we can remove every vertex of degree 3 that is adjacent to a vertex of degree 6. (In the mesh in Fig. 1, these are all but two.) Removing those redundant vertices gives an *external reduced representation* that is about half the size of the full representation (see §6 for further discussion and precise statistics). The internal representation remains the same, storing all vertices, because all redundant vertices are reconstructed when the external reduced representation is loaded.

degree	0	3	4	5	6	total
count	66	1853	485	475	680	3559
%	1	52*	13	13	19	100

level	0	1	2	3	4	5	6	total
count	58	26	47	45	87	288	1302	1853
%	3	1	3	2	5	16	70*	100

Table 2. Distribution of vertex degrees (top) and of vertices of degree 3 by refinement depth (bottom) for the mesh in Fig. 1. Vertices of degree 0 are boundary vertices. Vertices of degree 3 are by far the most frequent vertices. Vertices of degree 3 at the highest refinement depth are the most numerous.

Normalized representation. Our representation is not unique, but merely for trivial reasons, as in many representation schemes. A vertex can be given different 3-adic lattice coordinates $[a,b,m]$ if we allow common factors of 3 in a and b. Normalized 3-adic lattice coordinates restore uniqueness in the geometry. The orientation of the stars of vertices of degree 3 and 6 is not uniquely determined because those stars have rotational symmetry (see Fig. 6). Uniqueness in the topology is restored by using *normalized orientations*: in $\{0,1,2,3\}$ for degree 3 and in $\{0,1\}$ for degree 6 (that is, k mod 4 for degree 3 and k mod 2 for degree 6). The last source of non-uniqueness is trivial: vertices can be listed in arbitrary order. Sorting the external representation restores uniqueness. A sorted external representation with normalized lattice coordinates and normalized star orientations is called a *normalized external representation*. Our representation scheme does not require normalized representations because we normalize lattice coordinates before indexing the cloud. Nevertheless, normalized representations are useful for archiving and for comparing models. Indeed, two representations are equivalent when they define the same mesh and this happens exactly when their normalized external representations are the same.

5. Using the representation

Having described our representation, we now explain how to use it, that is, how to create a representation for the base mesh, how to update the representation during refinement, how to reconstruct the edges and faces of the mesh from the representation, and how to convert it to standard representations. Updating the representation and reconstructing the mesh require complete information about the stars of the vertices, the focus of our representation.

Representing the base mesh. A simple way to create the base mesh is to create the vertices at the center of the hexagons and then the vertices around each center along the 6 basic directions given by the powers of ω^2. This is easily done using lattice coordinates. Indeed, if c is the center of an hexagon, then the vertices around it are $w_k = c + \omega^k$ for $k = 0,2,4,6,8,10$. The center c has degree 3 and its star has orientation 0. The vertices w_k for $k = 0,4,8$ have degree 6 and orientation 0; the vertices w_k for $k = 2,6,10$ have degree 3 and orientation 2 (see Fig. 2). To make a grid of hexagons, note that $c + \omega^2 + \omega^4$ and $c + 1 + \omega^2$ are the centers of two adjacent hexagons (see Fig. 9).

Standard stars. For convenience, we precompute the standard stars. Fig. 10 shows $\text{star}(d,0)$ for each degree d; they are centered at the origin and at level 0. Table 3 gives the lattice coordinates of all vertices in these stars. We compute $\text{star}(d,1)$ from $\text{star}(d,0)$ by multiplying its vertices by $\rho\omega = \frac{1}{3}(1+\omega^2)$, the

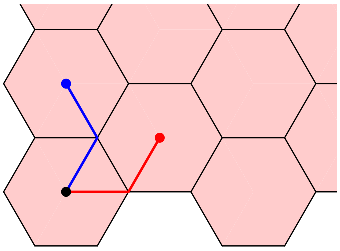


Fig. 9. Making a grid of hexagons for the base mesh. If c (black) is the center of an hexagon, then so are $c+\omega^2+\omega^4$ (blue) and $c+1+\omega^2$ (red).

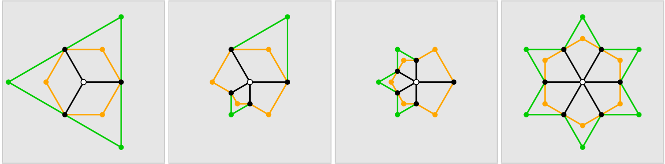


Fig. 10. Standard stars of each degree at orientation 0 as templates for reconstructing edges and faces around a vertex. Adjacent vertices shown in black, first opposite vertices in orange, second opposite vertices in green. Lattice coordinates for all vertices are given in Table 3.

degree	adjacent	opposite1	opposite2
3	$[1,0,0]$	$[0,1,0]$	$[0,2,0]$
	$[-1,1,0]$	$[-1,0,0]$	$[-2,0,0]$
	$[0,-1,0]$	$[1,-1,0]$	$[2,-2,0]$
4	$[1,0,0]$	$[0,1,0]$	$[0,2,0]$
	$[-1,1,0]$	$[-1,0,0]$	
	$[-1,-1,1]$	$[0,-2,1]$	$[0,-1,0]$
	$[1,-2,1]$	$[1,-1,0]$	
5	$[1,0,0]$	$[0,1,0]$	
	$[-1,2,1]$	$[-2,2,1]$	$[-1,1,0]$
	$[-2,1,1]$	$[-2,0,1]$	$[-1,0,0]$
	$[-1,-1,1]$	$[0,-2,1]$	$[0,-1,0]$
	$[1,-2,1]$	$[1,-1,0]$	
6	$[1,0,0]$	$[2,2,1]$	$[1,1,0]$
	$[0,1,0]$	$[-2,4,1]$	$[-1,2,0]$
	$[-1,1,0]$	$[-4,2,1]$	$[-2,1,0]$
	$[-1,0,0]$	$[-2,-2,1]$	$[-1,-1,0]$
	$[0,-1,0]$	$[2,-4,1]$	$[1,-2,0]$
	$[1,-1,0]$	$[4,-2,1]$	$[2,-1,0]$

Table 3. Lattice coordinates of the templates shown in Fig. 10. Standard stars are centered at the origin and at level 0.

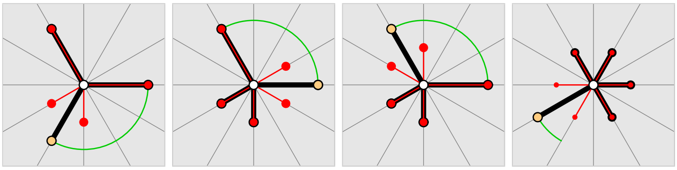


Fig. 11. Updating the orientation of the star of an old adjacent vertex u (white). Its old star in shown in black and its new star in red. The black edge joins u to the central vertex v in the old star of u. The orientation of u is updated according to the green arc.

barycenter of the triangle defined by the origin, 1, and ω^2. This reflects the effect of the local subdivision step on central edges: a rotation by 30° combined with a scaling by ρ. For $k \geq 2$, we compute $\text{star}(d,k)$ from $\text{star}(d,k-2)$ by multiplying its vertices by ω^2, a rotation by 60°. These computations are easily done in lattice coordinates using the fact that $\omega^4 = \omega^2 - 1$.

Finding vertex stars. The standard stars are templates for all stars: the star of a vertex v of degree d and orientation k is found by scaling $\text{star}(d,k)$ to the level of v and translating it to v. These operations are performed in lattice coordinates. More precisely, if $v = [a,b,m]$, then for each vertex $[a',b',m']$ of $\text{star}(d,k)$, the corresponding vertex w of the star of v is given by $w = [a,b,m] + [a',b',m' + \lfloor n/2 \rfloor]$, where n is the level of v. This is essentially adding fractions. The term $\lfloor n/2 \rfloor$ reflects the fact that it takes two subdivisions to scale a star by $\rho^2 = \frac{1}{3}$. Normalization of lattice coordinates is important here (and elsewhere): computing the lattice coordinates of w may involve fractions with greater denominators than those present in w (for instance, when v is at level $n \geq 2$ and w has depth 0). Hence the need for simplification.

Updating the representation. In each local subdivision operation (Fig. 5), we create new adjacent vertices and give the right orientation and scale to their stars, and we update the stars of the central vertex and of the old adjacent vertices, which become new opposite vertices. The old opposite vertices are not touched. Boundary vertices are never touched because they remain boundary vertices and have no topological data to update. In what follows, all orientations are computed modulo 12.

Consider a local subdivision operation on a central vertex v having level n and orientation k. After the subdivision, v has a new level and a new orientation; its degree remains 6. The new level of v is $n+1$, because the new central edges are scaled by ρ. The new orientation of v is $k+1$, because the new star is rotated 30° with respect to the old star. The new star of v can be computed directly, as explained above; there is no need to compute barycenters to find the new adjacent vertices.

Let w be a new vertex adjacent to v. Then w has degree 3 and is at level $n+1$ because it shares an edge with v. The j-th vertex in $\text{star}(6,0)$ has orientation $2j$. Therefore, if w is the j-th vertex in the new star of v, then the edge vw has orientation $2j+(k+1)$ and so the edge wv has the reverse orientation, $6+2j+(k+1)$; this is the orientation of w, because the orientation of a vertex of degree 3 is the same as the orientation of any of its edges.

Let u be an old vertex adjacent to v. After the subdivision, u becomes opposite to v and adjacent to two consecutive new vertices adjacent to v. Therefore, the degree of u increases by 1 because it loses one edge and gains two. The level of u remains the same, except when its old degree is 5, because then it loses its longest edge (see Fig. 11); in this case, the level of u is incremented. It remains to update the orientation of u to reflect its new star. Since the angles around v are all 60°, the angles in the old star of u can be only 90° or 120°, never 60°. Therefore, there are very few possibilities for the edge uv in the old star of u (see Fig. 11). If u is the j-th vertex in the old star of v, then vu has orientation $2j+k$ and so uv has the reverse orientation $k' = 6+2j+k$. To find the orientation of the new star u, just imagine rotating uv until it matches the edge that is horizontal in the corresponding standard star (see Fig. 6). More precisely, the orientation of the new star of u is $k'+4$ when the old degree of u is 3, $k' \pm 4$ when the degree is 4, and $k' + 1$ when the degree is 5 (see Fig. 11).

Reconstructing the mesh. To reconstruct the mesh from our representation, we need to find the edges and the faces of the mesh from the information about the vertices. To ensure consistency of the reconstructed edges and faces, we follow a standard approach in solid modeling and handle vertices as topological entities: only one instance of each vertex is present and adjacent elements refer to those instances. We rely on the cloud for this key task: it maps lattice coordinates to topological vertices.

To reconstruct the edges around a given vertex, we need to find the vertices adjacent to it. Those vertices are given by the star of the vertex, which is obtained by scaling and translating the appropriate standard star, as we have seen above. To reconstruct the faces around a given vertex, we need to find the vertices opposite to it (see Fig. 10). When the angle at the vertex is 90°, there is only one possible opposite vertex. When the angle at the vertex is 60° or 120°, there are two possible opposite vertices. To reconstruct the face in this case, we use the first opposite vertex, the one closest to the center, if it is present in the cloud; otherwise, we use the second opposite vertex, the one farthest from the center. Table 3 gives the lattice coordinates of all opposite vertices in star$(d,0)$; they need to be scaled and translated as before. Finally, the j-th face around a vertex v is given by the vertices v, a_j, o_j, a_{j+1}, where a_j is the j-th adjacent vertex and o_j is the j-th opposite vertex.

Reconstructing redundant vertices. After loading a reduced representation, we need to reconstruct the redundant vertices that have been omitted. In a single linear-time pass, we traverse the list of vertices ensuring that all the vertices adjacent to each vertex v of degree 6 are present in the cloud. Every such vertex w that is absent from the cloud is a vertex of degree 3 that has been omitted and needs to be added to the cloud. The lattice coordinates of w for indexing the cloud are known from the star of v. The level and orientation of w are found as in the update step described above: the level of w is that same as the level of v; the orientation of w is $6+2j+k$, if w is the j-th vertex in the star of v and k is the orientation of v.

Conversion. Our representation does not assign IDs to vertices: their lattice coordinates are effectively their IDs and the cloud directly supports this. Nevertheless, sequential numerical vertex IDs are convenient when one needs to process edges and faces without repetition, because edges and faces can be given standard representations using lists of vertex IDs, normalized by placing the smallest ID first. Those vertex lists can then be used to index hash tables for storing edges and faces without duplication. This scheme directly supports creating both an internal representation of the mesh using standard topological data structures [3] and an external representation of the mesh in a standard format that relies on vertex lists for faces, such as OBJ and OFF.

On the geometry side, the Cartesian coordinates of the vertices are not needed for subdivision, but they may be needed by the application using the mesh, typically to evaluate refinement criteria. They are also needed for creating OBJ and OFF files. Converting lattice coordinates $[a,b,m]$ to Cartesian coordinates (x,y) is simple: $\frac{1}{3^m}(a+b\omega^2) = \frac{1}{3^m}(a+b\frac{1+\sqrt{3}i}{2})$ gives $x = \frac{1}{3^m}(a+b\frac{1}{2})$ and $y = \frac{1}{3^m}(b\frac{\sqrt{3}}{2})$. Thus, Cartesian coordinates can be found as precisely as needed by the application, given the value of $\sqrt{3}$ to sufficient precision; standard double precision suffices for typical meshes.

6. Some statistics

We now report some statistics on how our representation of adaptive diamond-kite meshes behaves as a function of the refinement level. We consider two use cases: uniform meshes, where all vertices of degree 6 are refined to a given refinement level, and meshes refined adaptively around an implicit curve, as in Fig. 1, where vertices of degree 6 are refined when at least one of their edges crosses the curve. Uniform meshes are the densest possible meshes and give upper bounds for the sizes one can expect in applications. On the other hand, uniform meshes offer the most opportunities for data reduction. Meshes for implicit curves are typical of meshes refined around the boundary of a region of interest. In all experiments, the base mesh is the one shown in Fig. 12 (top left). As an indication of typical space requirements, we report sizes of external mesh representations as CSV, OBJ, and OFF files. All three formats record both the geometry and the topology of the mesh: CSV files describe the geometry by 3-adic lattice coordinates and the topology indirectly by vertex stars; OBJ and OFF files list the Cartesian coordinates of the vertices (to some precision) and the faces as lists of vertices IDs. The results appear in Tables 4–13.

Background. Planar meshes satisfy Euler's famous formula: $V - E + F = \chi$, where V is the number of vertices, E is the number of edges, F is the number of faces, and χ is a constant that depends only on the topology of the meshed region. Indeed, $\chi = C - H$, where C is the number of connected components and H is the number of holes. (Our base mesh has $\chi = 1$.) Therefore, χ remains constant when the mesh is refined (without breaking or merging) and so $E \sim V + F$ as the mesh grows. Quadrilateral meshes satisfy $4F = 2E - B$, where B is the number of boundary edges. The refinement of an adaptive diamond-kite mesh never changes its boundary and so B remains constant. Therefore, $E \sim 2F$ as the mesh grows and so $F \sim V$ and $E \sim 2V$.

Uniform meshes. Table 4 shows how the size of uniform meshes grows with the refinement level R. The relations between V, E, and F discussed above are confirmed, of course. More interestingly, these numbers grow geometrically with a ratio rapidly approaching 3. Indeed, let V_d be the number of internal vertices of degree d. Then, $\beta + 3V_3 + 4V_4 + 5V_5 + 6V_6 = 2E$, where β accounts for the boundary vertices. Since boundary vertices stop changing degrees at the early refinement levels, β is essentially a constant (indeed, $\beta \leq 6V_0$, where V_0 is the number of boundary vertices). By far, most refinement happens in the interior of the mesh, which is composed of vertices of degree 3 and 6 (see Fig. 12). Therefore, we can ignore all other vertices (see Table 5) and get $3V_3 + 6V_6 \sim 2E \sim 4V \sim 4(V_3 + V_6)$. Thus, $V_3 \sim 2V_6$ and so $V \sim 3V_6$. Finally, during refinement all vertices

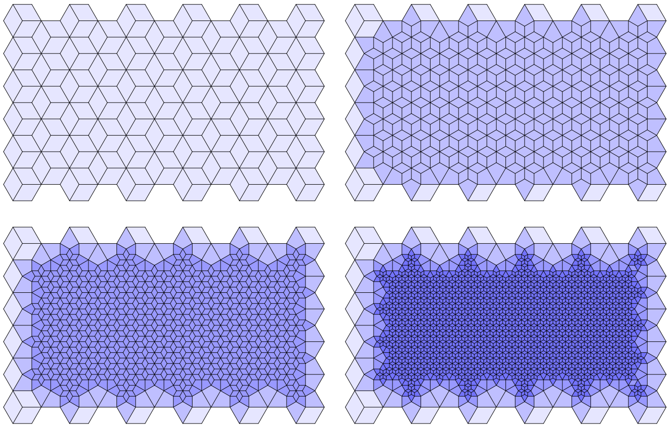


Fig. 12. Uniform meshes refined to levels 0 to 3.

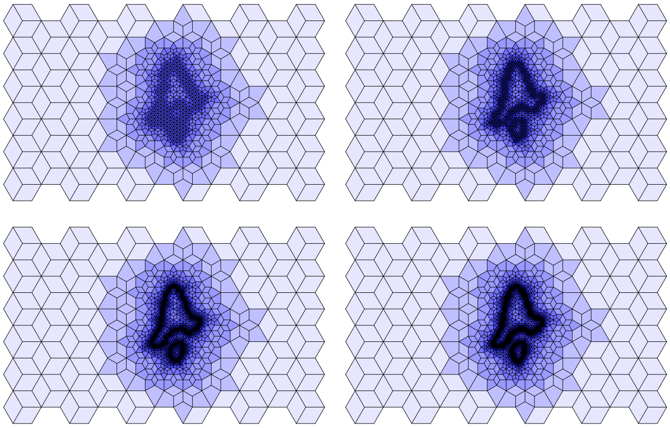


Fig. 13. Meshes for implicit curve refined to levels 4 to 7.

of degree 6 generate six new vertices of degree 3 and all the old vertices degree 3 become new vertices of degree 6, and so we have $V_3^{(n+1)} \sim 6V_6^{(n)} \sim 3V_3^{(n)}$, which explains the geometric growth with ratio approaching 3 since $2V \sim 6V_6 \sim 3V_3$.

Table 6 shows the sizes in bytes of external mesh representations as CSV, OBJ, and OFF files. CSV files are typically less than 30% of the size of OBJ and OFF files. Reduced representations are typically 33% smaller than full representations, as predicted, because most vertices have degree 6 and $V \sim 3V_6$. Table 7 shows the sizes in bytes of these files after compression with ‘bzip2 –9’ (the results of ‘gzip –9’ are quite similar and have been omitted). Compressed CSV files are typically about 20% of the size of compressed OBJ and OFF files. Compressed reduced representations are typically about 30% of compressed full representations.

As the mesh is refined, all vertices are created inside the region defined by the base mesh. Therefore, their Cartesian coordinates remain bounded and so the numerators a, b in lattice coordinates must grow to compensate for the denominator 3^m. Table 8 shows how the size of lattice coordinates grows. We see the same geometric growth ratio of 3, except that it happens every other step because the shortest mesh edges reduce by $\rho = \frac{1}{\sqrt{3}}$ on refinement.

Implicit curve. Tables 9–13 show the corresponding statistics for adaptive diamond-kite meshes around an implicit curve (see Fig. 13). Table 9 shows how the sizes of those meshes grow with the refinement level R. The numbers grow geometrically as before but now at a ratio less than 2. Table 10 shows that the vertices of degree 3 are still the majority, comprising more than half of all vertices. Tables 11 and 12 show that CSV files are about 30% of the size of OBJ and OFF files. Reduced representations are almost 50% smaller than full representations. Compressed CSV files are typically less than 30% of the size of compressed OBJ and OFF files. Compressed reduced representations are typically about 50% of compressed full representations. Table 13 shows that the size of lattice coordinates grows at the same geometric growth ratio of 3, every other step, just like uniform meshes, as expected.

7. Comparison with topological data structures

We now discuss and compare our representation with standard topological data structures and some variations [3, 9]. In the context of data structures, a *reference* is either a pointer or an integer that indexes an array; we assume it takes 4 bytes.

The simplest way to represent the topology of an adaptive diamond-kite mesh is the one used in OBJ and OFF files: a list of faces and their vertices. This takes 4 references per face and so 4 references per vertex (recall that $F \sim V$ and $E \sim 2V$). Including face adjacency relations requires 4 additional references per vertex. Standard topological data structures for a planar mesh are more complete and so use many more references per vertex [3]:

data structure	total references	per vertex
winged-edge	$V + 8E + F$	18
dcel	$V + 6E + F$	14
half-edge	$V + 10E + F$	22
quad-edge	$V + 8E + F$	18

The star-vertex data structure [9] uses about $2\delta + 2$ references per vertex, where δ is the average vertex degree in the mesh. Adaptive diamond-kite meshes have $\delta \approx 4$. Therefore, star-vertex uses about 10 references per vertex.

The memory used by those data structures is much larger than ours. In its simplest form, our representation uses 3 bytes per vertex for the topological data d, k, n. Given the narrow range of these fields, we can use $3 + 4 + 5 = 12$ bits per vertex for this data since the degree is at most 6, the orientation is at most 11, and the level is most probably less than 32. In both cases, our representation uses less than 1 reference per vertex.

Representing the geometry of the mesh with single-precision floating-point numbers and with 3-adic lattice coordinates takes about the same amount of space. Indeed, with single-precision floating-point numbers it takes 64 bits per vertex, and we can comfortably use 24 bits for a, b and 8 bits for m and so 56 bits per vertex. This is a fair comparison, since single-precision numbers have a mantissa of 23 bits.

In summary, our representation takes 68 bits per vertex: 56 bits for the geometric data a, b, m, which is the key to query the cloud, and 12 bits for the topological data d, k, n, which is the value stored in the cloud for that key.

To ensure good performance of our representation, we assume that the cloud is a well-implemented hash table that answers queries in expected constant time and uses memory efficiently with high occupancy and low bookkeeping. Then all local operations that query or update our representation take expected constant time. Retrieving adjacent elements in topological data structures typically requires many pointer indirections, which may slow down local queries and updates.

8. Conclusion

The conciseness of our vertex-centric representation for adaptive diamond-kite meshes is achieved by (1) representing vertex coordinates with integers that can index a hash table to store vertices as topological entities and (2) exploiting the rigidity of vertex stars to avoid representing explicitly any topological relations. Even so, all topological elements and relations are reconstructed in expected constant time per element. We rely on a good hash table for this performance. Precomputing standard stars simplifies the key tasks.

Our representation can be easily extended to general diamond-kite meshes by replacing the degree with a type corresponding to the 11 possible arrangements of angles around a vertex given in Table 1. However, we do not know how to extend the local subdivision operation to handle such general meshes. Nevertheless, our representation can be easily adapted to diamond-kite meshes that use different refinement procedures. The appendix contains a brief description of how our vertex-centric approach provides a framework for representing Fathauer's kite fractals [6].

Acknowledgements. I thank David Eppstein for generously sharing his proof-of-concept Python implementation of adaptive diamond-kite meshes; Craig Kaplan for pointing me to the work of Robert Fathauer on kite fractals; Asla Medeiros e Sá for showing me Eppstein's paper and hearing my thoughts about it; and an anonymous referee for suggesting it might be possible to remove some internal vertices, which led me to reduced representations. This research was done in the Visgraf Computer Graphics laboratory at IMPA in Rio de Janeiro, Brazil. Visgraf is supported by the funding agencies FINEP, CNPq, and FAPERJ, and also by gifts from IBM Brasil, Microsoft, NVIDIA, and other companies.

References

[1] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin. Quad-mesh generation and processing: A survey. *Computer Graphics Forum*, 32(6):51–76, 2013.

[2] L. H. de Figueiredo. A vertex-centric representation for adaptive diamond-kite meshes. Code at github.com/lhf/dk, 2024.

[3] L. De Floriani and A. Hui. Shape representations based on simplicial and cell complexes. In *Eurographics 2007 State of the Art Reports*, pages 63–87. Eurographics, 2007.

[4] R. Diaz, M. Dreux, H. Lopes, and T. Lewiner. A simple compression of tri-quad meshes with handles. In *Full Papers Proceedings of WSCG 2010*, pages 205–212, 2010.

[5] D. Eppstein. Diamond-kite adaptive quadrilateral meshing. *Engineering with Computers*, 30(2):223–235, 2014.

[6] R. W. Fathauer. Fractal tilings based on kite- and dart-shaped prototiles. *Computers & Graphics*, 25(2):323–331, 2001.

[7] R. Fellegara, K. Weiss, and L. De Floriani. The stellar decomposition: A compact representation for simplicial complexes and beyond. *Computers & Graphics*, 98:322–343, 2021.

[8] J. Fuentes-Sepúlveda, G. Navarro, and D. Seco. Navigating planar topologies in near-optimal space and time. *Computational Geometry*, 109:101922, 2023.

[9] M. Kallmann and D. Thalmann. Star-vertices: A compact representation for planar meshes with adjacency information. *Journal of Graphics Tools*, 6(1):7–18, 2001.

[10] D. King, J. Rossignac, and A. Szymczak. Connectivity compression for irregular quadrilateral meshes, 2000. arXiv:cs/0005005.

[11] G. Navarro. *Compact Data Structures: A Practical Approach*. Cambridge University Press, 2016.

[12] J. E. Soto Sánchez, T. Weyrich, A. Medeiros e Sá, and L. H. de Figueiredo. An integer representation for periodic tilings of the plane by regular polygons. *Computers & Graphics*, 95:69–80, 2021.

[13] B. Von Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. *Computer Graphics*, 21(4):103–110, 1987. (Proc. SIGGRAPH '87).

STATISTICS FOR UNIFORM MESHES

R	V	E	F	E/V	F/V	V growth
0	217	399	183	1.839	0.843	1.000
1	487	939	453	1.928	0.930	2.244
2	1135	2235	1101	1.969	0.970	2.331
3	2767	5499	2733	1.987	0.988	2.438
4	7075	14115	7041	1.995	0.995	2.557
5	18979	37923	18945	1.998	0.998	2.683
6	52891	105747	52857	1.999	0.999	2.787
7	151483	302931	151449	2.000	1.000	2.864
8	441763	883491	441729	2.000	1.000	2.916
9	1302931	2605827	1302897	2.000	1.000	2.949

Table 4. Sizes and growth of uniform meshes.

R	0	3	4	5	6	total	V_3/V
0	66	106	0	0	45	217	0.49
1	66	272	16	25	108	487	0.56
2	66	650	72	75	272	1135	0.57
3	66	1634	182	167	718	2767	0.59
4	66	4310	404	311	1984	7075	0.61
5	66	11906	784	571	5652	18979	0.63
6	66	33914	1460	1019	16432	52891	0.64
7	66	98594	2640	1803	48380	151483	0.65
8	66	290282	4692	3195	143528	441763	0.66
9	66	861170	8308	5635	427752	1302931	0.66

Table 5. Distribution of degrees for uniform meshes.

R	CSV	OBJ	OFF	CSV/OBJ	CSV/OFF	reduced	red/full
0	2751	7680	7239	0.36	0.38	1460	0.53
1	6466	18434	17450	0.35	0.37	2767	0.43
2	15321	51210	48927	0.30	0.31	6461	0.42
3	38528	133271	127724	0.29	0.30	15330	0.40
4	99571	365131	350968	0.27	0.28	38495	0.39
5	284920	1023179	985202	0.28	0.29	99636	0.35
6	812611	2978332	2872530	0.27	0.28	284881	0.35
7	2398081	8803735	8500744	0.27	0.28	812775	0.34
8	7064084	26680004	25796453	0.26	0.27	2397658	0.34
9	22085931	80702020	78096136	0.27	0.28	7064290	0.32

Table 6. Size of files in bytes.

R	CSV	OBJ	OFF	CSV/OBJ	CSV/OFF	reduced	red/full
0	472	1654	1656	0.29	0.29	282	0.60
1	1214	3901	3914	0.31	0.31	555	0.46
2	2626	9098	9132	0.29	0.29	1309	0.50
3	6527	23825	24119	0.27	0.27	2828	0.43
4	15804	62244	62410	0.25	0.25	6902	0.44
5	45346	186156	182900	0.24	0.25	16664	0.37
6	122356	580741	580689	0.21	0.21	46894	0.38
7	403829	1922938	1936033	0.21	0.21	136289	0.34
8	1177249	6245451	6268366	0.19	0.19	407279	0.35
9	3988817	20187852	20281873	0.20	0.20	1233642	0.31

Table 7. Size of bzip2-compressed files in bytes.

R	min	max	m	growth
0	−6	16	0	1.000
1	−11	44	1	2.750
2	−11	44	1	1.000
3	−29	128	2	2.909
4	−29	128	2	1.000
5	−83	380	3	2.969
6	−83	380	3	1.000
7	−245	1136	4	2.989
8	−245	1136	4	1.000
9	−731	3404	5	2.996

Table 8. Size of lattice coordinates in uniform meshes.

STATISTICS FOR IMPLICIT CURVE

R	V	E	F	E/V	F/V	V growth
0	217	399	183	1.839	0.843	1.000
1	277	519	243	1.874	0.877	1.276
2	379	723	345	1.908	0.910	1.368
3	649	1263	615	1.946	0.948	1.712
4	1159	2283	1125	1.970	0.971	1.786
5	1981	3927	1947	1.982	0.983	1.709
6	3559	7083	3525	1.990	0.990	1.797
7	6247	12459	6213	1.994	0.995	1.755
8	10759	21483	10725	1.997	0.997	1.722
9	18949	37863	18915	1.998	0.998	1.761

Table 9. Sizes and growth of implicit meshes.

R	0	3	4	5	6	total	V_3/V
0	66	106	0	0	45	217	0.49
1	66	134	14	9	54	277	0.48
2	66	185	33	22	73	379	0.49
3	66	332	66	56	129	649	0.51
4	66	614	124	114	241	1159	0.53
5	66	1030	252	254	379	1981	0.52
6	66	1853	485	475	680	3559	0.52
7	66	3242	888	878	1173	6247	0.52
8	66	5543	1593	1589	1968	10759	0.52
9	66	9771	2825	2821	3466	18949	0.52

Table 10. Distribution of degrees for implicit meshes.

R	CSV	OBJ	OFF	CSV/OBJ	CSV/OFF	reduced	red/full
0	2751	7680	7239	0.36	0.38	1460	0.53
1	3595	10062	9498	0.36	0.38	1851	0.51
2	5029	15086	14318	0.33	0.35	2561	0.51
3	8830	27641	26333	0.32	0.34	4270	0.48
4	16035	53228	50896	0.30	0.32	7489	0.47
5	29047	96252	92275	0.30	0.31	13236	0.46
6	54193	180662	173528	0.30	0.31	25191	0.46
7	97540	324406	311897	0.30	0.31	45930	0.47
8	170543	569255	547718	0.30	0.31	81715	0.48
9	314507	1038938	1001021	0.30	0.31	145761	0.46

Table 11. Size of files in bytes.

R	CSV	OBJ	OFF	CSV/OBJ	CSV/OFF	reduced	red/full
0	472	1654	1656	0.29	0.29	282	0.60
1	723	2335	2318	0.31	0.31	395	0.55
2	1045	3200	3187	0.33	0.33	600	0.57
3	1838	5769	5767	0.32	0.32	926	0.50
4	3168	10192	10227	0.31	0.31	1679	0.53
5	5702	18610	18675	0.31	0.31	2765	0.48
6	9783	33530	33675	0.29	0.29	5169	0.53
7	18313	62125	62407	0.29	0.29	8671	0.47
8	30900	106479	107177	0.29	0.29	15873	0.51
9	58458	212883	211461	0.27	0.28	27430	0.47

Table 12. Size of bzip2-compressed files in bytes.

R	min	max	m	growth
0	−6	16	0	1.000
1	−6	29	1	1.813
2	−6	29	1	1.000
3	−6	71	2	2.448
4	−6	74	2	1.042
5	−6	203	3	2.743
6	−6	206	3	1.015
7	−6	602	4	2.922
8	−6	602	4	1.000
9	−6	1796	5	2.983

Table 13. Size of lattice coordinates in implicit meshes.

Appendix.
A vertex-centric representation for Fathauer's kite fractals

Our vertex-centric approach provides a framework for representing diamond-kite meshes that use different refinement procedures. One interesting such family of meshes is Fathauer's kite fractals [6] (see Fig. 14). The core of our representation for those meshes is the same: 3-adic lattice coordinates, topology represented by the type, orientation, and scale of vertex stars, the cloud for storing vertices, standard stars as templates. However, all details are different: refinement happens only at the boundary, holes appear during refinement, and vertex stars are completely different. Here is a brief description of the details.

Stars. The standard vertex stars in Fathauer's kite fractals are shown in Fig. 15. The lattice coordinates of adjacent and opposite vertices in these stars are easily found. Vertices of type 20, 31, 32 are boundary vertices. Vertices of type 41, 42, 43, 50, 60 are internal vertices. The first digit of the type gives the degree of the vertex.

Base mesh. The base mesh is formed by six kites around the origin (see Fig. 14, top left). It contains boundary vertices of type 20 and 31, and a single internal vertex of type 60. Vertices of type 32 first appear at refinement depth 1.

Refinement. Fathauer [6] briefly explains the refinement procedure geometrically, but it is easy to explain it precisely using vertex types. At each refinement step, the boundary vertices are refined and become internal vertices, thus expanding the boundary of the mesh: vertices of type 20 are refined to type 50, vertices of type 31 are refined to type 41, vertices of type 32 are refined to type 42. The refinement is performed by simply updating the type of the vertices; the standard stars have been selected so that the orientation remains the same during refinement. The new vertices created at that step are given proper type, orientation, and level, following an analysis similar to that in §5. After refinement depth 5, collisions start to appear as the boundary expands, thus creating hexagonal holes in the mesh (see Fig. 16). The new vertices created when a vertex of type 20 becomes of type 50 are checked to exist in the cloud. If they do, then the boundary has collided with itself and the new vertices are updated to type 32 and 43 with the proper orientation.

Conclusion. Our vertex-centric framework for representing adaptive diamond-kite meshes works also for representing Fathauer's kite fractals, which are adaptive kite meshes. The refinement is translated into vertex refinements, which are essentially changes of vertex type. The appearance of collisions was surprising but added interest; they are easily handled by our framework. Again, the cloud is instrumental to simplify the code. Despite having simpler faces, Fathauer's kite fractals have a richer topology (more vertex types and the continuous appearance of holes) and so are somewhat more complicated to represent in our framework than Eppstein's meshes. Nevertheless, our framework provides a reliable guide for the whole task. Our code for kite fractals using our vertex-centric framework is publicly available [2].

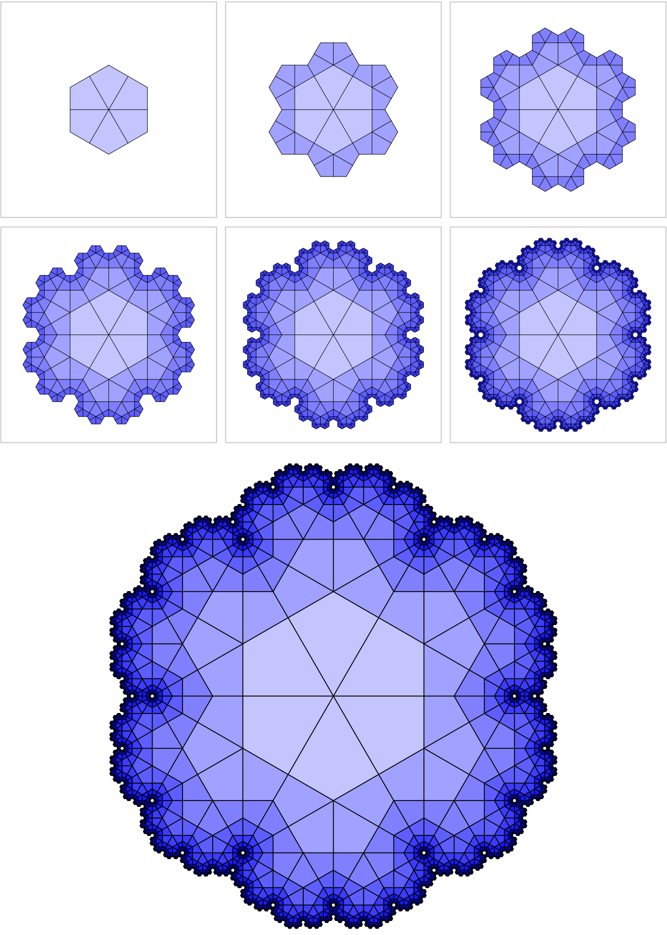


Fig. 14. A Fathauer kite fractal refined to depths 0 to 5 and 7.

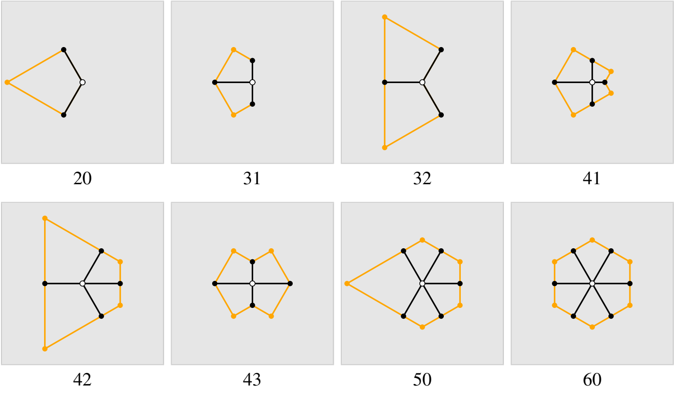


Fig. 15. The possible vertex stars in Fathauer's kite fractals at orientation 0. Adjacent vertices and edges in black; opposite vertices and edges in orange.

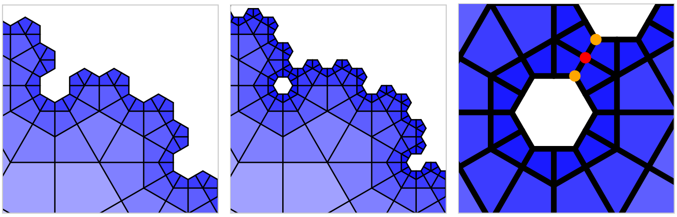


Fig. 16. No holes at depth 4 (left). Holes start to appear at depth 5 (middle). Vertices of type 20 and 31 become of type 32 and 43 (right) on collision.