Straight Skeletons of Three-Dimensional Polyhedra*

Gill Barequet[1], David Eppstein[2], Michael T. Goodrich[2], and Amir Vaxman[1]

[1] Dept. of Computer Science
Technion—Israel Institute of Technology
Haifa 32000, Israel
{barequet,avaxman}@cs.technion.ac.il
[2] Computer Science Department
Univ. of California, Irvine
{eppstein,goodrich}@ics.uci.edu

Abstract. We study the straight skeleton of polyhedra in 3D. We first show that the skeleton of voxel-based polyhedra may be constructed by an algorithm taking constant time per voxel. We also describe a more complex algorithm for skeletons of voxel polyhedra, which takes time proportional to the surface-area of the skeleton rather than the volume of the polyhedron. We also show that any n-vertex axis-parallel polyhedron has a straight skeleton with $O(n^2)$ features. We provide algorithms for constructing the skeleton, which run in $O(\min(n^2 \log n, k \log^{O(1)} n))$ time, where k is the output complexity. Next, we show that the straight skeleton of a general nonconvex polyhedron has an ambiguity, suggesting a consistent method to resolve it. We prove that the skeleton of a general polyhedron has a superquadratic complexity in the worst case. Finally, we report on an implementation of an algorithm for the general case.

1 Introduction

The straight skeleton is a geometric construction that reduces two-dimensional shapes—polygons—to one-dimensional sets of segments approximating the same shape. It is defined in terms of an offset process in which edges move inward, remaining straight and meeting at vertices. When a vertex meets an offset edge, the process continues within the two pieces so formed. The straight segments traced out by vertices during this process define the skeleton. Introduced by Aichholzer *et al.* [1,2], the two-dimensional straight skeleton has since found many applications, e.g., surface folding [9], offset curve construction [13], interpolation of surfaces in three dimensions from cross sections [3], automated interpretation of geographic data [15], polygon decomposition [21], etc. The straight skeleton is more complex to compute than other types of skeleton [6,13], but its piecewise-linear form offers many advantages. The best known alternative, the medial axis [5], consists of both linear and quadratic curve segments.

* Work on this paper by the first and fourth authors has been supported in part by a French-Israeli Research Cooperation Grant 3-3413.

D. Halperin and K. Mehlhorn (Eds.): ESA 2008, LNCS 5193, pp. 148–160, 2008.

It is natural, then, to try to develop algorithms for skeleton construction of a polyhedron in 3D. The most well-known type of 3D skeleton, the medial axis, has found applications, e.g., in mesh generation [17] and surface reconstruction [4]. Unlike its 2D counterpart, the 3D medial axis can be quite complex, both combinatorially and geometrically. Thus, we would like an alternative way to characterize the shape of 3D polyhedra using a simpler type of 2D skeleton.

1.1 Related Prior Work

We are not aware of any prior work on 3D straight skeletons, other than Demaine *et al.* [8], who give the basic properties of 3D straight skeletons, but do not study them in detail w.r.t. their algorithmic, combinatorial, or geometric properties.

Held [16] showed that in the worst case, the complexity of the medial axis of a convex polyhedron of complexity n is $\Omega(n^2)$, which implies a similar bound for the 3D straight skeleton. Perhaps the most relevant prior work is on shape characterization using the 3D medial axis, defined from a 3D polyhedron as the Voronoi diagram of the set of faces, edges, and vertices of the polyhedron. The best known upper bound for its combinatorial complexity is $O(n^{3+\varepsilon})$ [18].

Because of these drawbacks, a number of researchers have studied algorithms for approximating 3D medial axes. Sherbrooke *et al.* [20] give an algorithm that traces out the curved edges of the 3D medial skeleton. Culver *et al.* [7] use exact arithmetic to compute a representation of a 3D medial axis. In both cases, the running time depends on both the combinatorial and geometric complexity of the medial axis. Foskey *et al.* [14] construct an approximate medial axis using a voxel-based approach that runs in time $O(nV)$, where n is the number of features of the input polyhedron and V is the volume of the voxel mesh that contains it. Sheehy *et al.* [19] use instead the 3D Delaunay triangulation of a cloud of points on the surface of the input polyhedron to approximate 3D medial axis. Likewise, Dey and Zhao [10] study the 3D medial axis as a subcomplex of the Voronoi diagram of a sampling of points approximating the input polyhedron.

1.2 Our Results

- We study the straight skeleton of orthogonal polyhedra formed as unions of voxels. We analyze how the skeleton may intersect each voxel, and describe a suitable a simple voxel-sweeping algorithm taking constant time per voxel.
- We give a more complex algorithm for skeletons of voxel polyhedra, which, rather than taking time proportional to the total volume, takes time proportional to the the number of voxels it intersects.
- We show that any n-vertex axis-parallel polyhedron has a skeleton with $O(n^2)$ features. We provide two algorithms for computing it, resulting in a runtime of $O(\min(n^2 \log n, k \log^{O(1)} n))$, where k is the output complexity.
- We discuss the ambiguity in defining skeletons for general polyhedra and suggest a consistent method for resolving it. We show that for a general polyhedron, the straight skeleton can have superquadratic complexity. We also describe an algorithm for computing the skeleton in the general case.

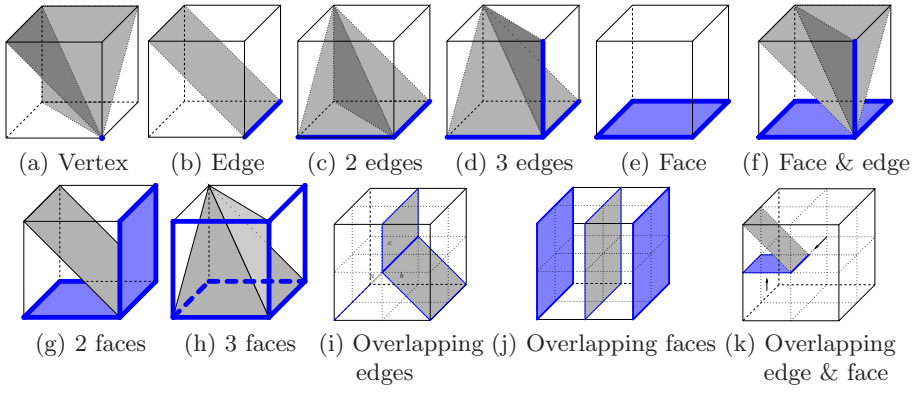


Fig. 1. Cases of straight skeleton within a subvoxel (a-h) or voxel (i-k)

2 Voxel Polyhedra

In this section we consider the case in which the polyhedron is a polycube, that is, a rectilinear polyhedron all of whose vertices have integer coordinates. The "cubes" making up the polyhedron are also called voxels. For voxels, and more generally for orthogonal polyhedra, the straight skeleton is a superset of the L_∞ Voronoi diagram. Due to this relationship, the straight skeleton is significantly easier to compute for orthogonal inputs than in the general case.

As in the general case, the straight skeleton of a polycube can be modeled by offsetting the boundary of the polycube inward, and tracing the movement of the boundary. During this sweep, the boundary forms a moving front whose features are faces, edges, and vertices. An edge can be either convex or concave, while a vertex can be convex, concave, or a saddle. In the course of this process, features may disappear or appear.

The sweep starts at time 0, when the front is the boundary of the polycube. In the first time unit we process all the voxels adjacent to the boundary. In the ith round ($i \geq 1$) we process all the voxels adjacent to voxels processed in the $(i-1)$st round, that have never been processed before. Processing a voxel means the computation of the piece of the skeleton lying within the voxel. During this process, the polycube is shrunk, and may be broken into several components. The process continues for every piece separately until it vanishes.

2.1 A Volume Proportional-Time Algorithm

Theorem 1. *The combinatorial complexity of the straight skeleton of a polycube of volume V is $O(V)$. The skeleton can be computed in $O(V)$ time.*

Proof. The claims follow from the fact that the complexity of the skeleton within every voxel (or, more precisely, within every 1/8-voxel), as well as the time needed to compute it during the sweep, is $O(1)$. Fig. 1 illustrates the different cases. The full details are given in the full version of the paper. □

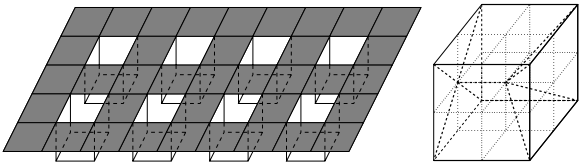


Fig. 2. A polycube of volume V whose skeleton has complexity $\Theta(V)$

This algorithm is worst-case optimal, since in the worst case the complexity of the skeleton of a polycube made of V voxels is $\Theta(V)$. One such example, shown in Fig. 2(a), is made of a flat layer of cubes (not shown), with a grid of supporting "legs," each a single cube. The number of legs is about 1/5 of the total number of voxels. The skeleton of this object has features within every leg, see Fig. 2(b) (the bottom of a leg corresponds to the right side of the figure).

2.2 Output-Sensitive Voxel Sweep

The straight skeleton of a polycube, as constructed by the previous algorithm, contains features within some voxels, but other voxels may not participate in the skeleton; nevertheless, the algorithm must consider all voxels and pay in its running time for them. In this section we outline a more efficient algorithm that computes the straight skeleton in time proportional only to the number of voxels containing skeleton features, or equivalently, in time proportional to the *surface area* of the straight skeleton rather than its volume. Necessarily, we assume that the input polycube is provided as a space-efficient boundary representation rather than as a set of voxels, for otherwise simply scanning the input would take more time than we wish to spend.

Our algorithm consists of an outer loop, in which we advance the moving front of the polycube boundary one time step at a time, and an inner loop, in which we capture all features of the straight skeleton formed in that time step. During the algorithm, we maintain at each step a representation of the moving front, as a collection of polygons having orthogonal and diagonal edges. As long as each operation performed in the inner and outer loops of the algorithm can be charged against straight skeleton output features, the total time will be proportional to the output size.

In order to avoid the randomization needed for hashing, several steps of our algorithm will use as a data structure a direct-addressed lookup table, which we summarize in the following lemma:

Lemma 1. *In time proportional to the boundary of an input polycube, we may initialize a data structure that can repeatedly process a collection of objects, indexed by integers within the range of coordinates of the polycube vertices, and produce as output a graph, in which the vertices are sets of objects that have equal indices and the edges are pairs of sets with index values that differ by one. The time per operation is proportional to the number of objects given as input.*

In each step of the outer loop of the algorithm, we perform the following:

1. Advance each face of the wavefront one unit inward. In this advancement process, we may detect events in which a wavefront edge shrinks to a point, forming a straight skeleton vertex. However, events involving pairs of features that are near in space but far apart on the wavefront may remain undetected. Thus, after this step, the wavefront may include overlapping pairs of coplanar oppositely-moving faces.
2. For each plane containing faces of the new wavefront boundary, detect pairs of faces that overlap within that plane, and find the features in which two overlapping face edges intersect or in which a vertex of one face lies in the interior of another face. (Details are provided in the full version of the paper.)
3. In the inner loop of the algorithm, propagate straight skeleton features within each face of the wavefront from the points detected in the previous step to the rest of the face. If two faces overlap in a single plane, the previous step will have found some of the points at which they form skeleton vertices, but the entire overlap region will form a face of the skeleton. We propagate outward from the detected intersection points using DFS, voxel by voxel, to determine the skeleton features contained within the overlap region.

In summary, we have:

Theorem 2. *One can compute the straight skeleton of a polycube in time proportional to its surface area.*

3 Orthogonal Polyhedra

3.1 Definition

We consider here the more general *orthogonal polyhedra*, in which all faces are parallel to two of the coordinate axes. As in the 2D case, we define the straight skeleton of an orthogonal polyhedron P by a continuous shrinking process in which a sequence of nested "offset surfaces" are formed, starting from the boundary of the polyhedron, with each face moving inward at a constant speed. At time t in this process, the offset surface P_t for P consists of the set of points at L_∞ distance exactly t from the boundary of P. For almost all values of t, P_t will be a polyhedron, but at some time steps P_t may have a non-manifold topology, possibly including flat sheets of surface that do not bound any interior region. When this happens, the evolution of the surface undergoes sudden discontinuous changes, as these surfaces vanish at time steps after t in a discontinuous way. More precisely, we define a *degenerate point* of P_t to be a point p that is on the boundary of P_t, s.t., for some δ, and all $\varepsilon > 0$, $P_{t+\varepsilon}$ does not contain any point within distance δ of p.

At each step in the shrinking process, we imagine the surface of P_t as *decorated* with *seams* left over when sheets of degenerate points occur. Specifically, suppose that P contains two disjoint xy-parallel faces at the same z-height; then, as we shrink P, the corresponding faces of P_t may grow toward each other. When they

meet, they leave a seam between them. Seams can also occur when two parts of the same nonconvex face grow toward and meet each other. After a seam forms, it remains on the face of P_t on which it formed, orthogonal to the position at which it originally formed.

We define the straight skeleton of P as the union of three sets: 1. Points that, for some t, belong to an edge or vertex of P_t; 2. Degenerate points for P_t for some t; and 3. Points that, for some t, belong to a seam of P_t.

3.2 Complexity Bounds

As each face has at least one boundary edge, and each edge has at least one vertex, we may bound the complexity of the straight skeleton by bounding the number of its vertices. Each vertex corresponds to an *event*, that is, a point p (the location of the vertex), the time t for which p belongs to the boundary of P_t, and the set of features of $P_{t-\varepsilon}$ near p for small values of ε that contribute to the event. We may classify events into six types.

Concave-vertex events, in which one of the features of $P_{t-\varepsilon}$ involved in the event is a *concave vertex*: that is, a vertex of $P_{t-\varepsilon}$ s.t. seven of the eight quadrants surrounding that vertex lie within $P_{t-\varepsilon}$. In such an event, this vertex must collide against some oppositely-moving feature of P_t.

Reflex-reflex events are not concave-vertex events, but these events involve the collision between two components of boundary of $P_{t-\varepsilon}$ that prior to the event are far from each other as measured in geodesic distance around the boundary, both of which include a reflex edge. These components may either be a reflex edge, or a vertex that has a reflex edge within its neighborhood.

Reflex-seam events are not either of the above two types, but they involve the collision between two different components of boundary of $P_{t-\varepsilon}$, one of which includes a reflex edge. The other component must be a seam edge or vertex, because it is not possible for a reflex edge to collide with a convex edge of $P_{t-\varepsilon}$ unless both edges are part of a single boundary component.

Seam-seam events in which vertices or edges on two seams, on oppositely oriented parallel faces of $P_{t-\varepsilon}$, collide with each other.

Seam-face events in which a seam vertex on one face of $P_{t-\varepsilon}$ collides with a point on an oppositely oriented face that does not belong to a seam.

Single-component events in which the boundary points near p in $P_{t-\varepsilon}$ form a single connected subset.

Theorem 3. *The straight skeleton of an n-vertex orthogonal polyhedron has complexity $O(n^2)$.*

Proof. (Sketch) We count the events of each different type. There are $O(n)$ concave-vertex and seam-face events, while there are $O(n^2)$ reflex-reflex, reflex-seam, and seam-seam events. Single-component events can be charged against the events of other types. Each event contributes a constant amount of skeletal features. More details are given in the full version of the paper. □

3.3 Algorithms

Again, we view the skeleton as generated by a moving surface that changes at discrete events. It is easy to fully process an event in constant time, so the problem reduces to determining efficiently the sequence of events, and distinguishing actual events from false events. To this aim we provide two algorithms.

Theorem 4. *There is a constant c, s.t. the skeleton of an n-vertex orthogonal polyhedron with k skeletal features may be constructed in time $O(k \log^c n)$.*

Proof. Each event in our classification (except single-component events, which may be handled by an event queue) is generated by the interaction of two features of the moving surface P_t. To generate these events, ordered by the time at which they occur, we use a data structure of Eppstein [11,12] for maintaining a set of items and finding the pair of items minimizing some binary function $f(x, y)$—the time at which an event is generated by the interaction of items x and y ($+\infty$ if there is no interaction). The data structure reduces this problem (with polylogarithmic overhead) to a simpler problem: maintain a dynamic set X of items, and answer queries asking for the first interaction between an item $x \in X$ and a query item y. We need separate first-interaction data structures of this type for edge-edge, vertex-face, and face-vertex interactions. In the full version of the paper we provide the implementation details of these data structures. □

A simpler algorithm is worst-case (rather than output-sensitive) optimal.

Theorem 5. *The straight skeleton of an orthogonal polyhedron with n vertices and k straight skeleton features may be constructed in time $O(n^2 \log n)$.*

Proof. For each pair of objects that may interact (features of the input polyhedron P or of the 2D straight skeletons S_Π in each face plane Π), we compute the time of interaction. We process the pairs of objects by the order of these times; whenever we process a pair (x, y), we consult an additional data structure to determine whether the pair causes an event or whether the event that they might have caused has been blocked by some other features of the skeleton.

To test whether an edge-edge pair causes an event, we maintain a binary search tree for each edge, representing the family of segments into which the line containing that edge (translated according to the motion of the surface P_t) has been subdivided in the current state of the surface P_t. An edge-edge pair causes an event if the point at which the event would occur currently belongs to line segments from the lines of both edges, which may be tested in logarithmic time.

To test whether a vertex-face pair causes an event, we check whether the vertex still exists at the time of the event, and then perform a point location query to locate the point in S_Π at which it would collide with a face belonging to Π. The collision occurs if the orthogonal distance within Π from this point to the nearest face is smaller than the time at which the collision would occur. We do not need to check whether other features of the skeleton might have blocked features of S_Π from belonging to the boundary of P_t, for if they did they would also have led to an earlier vertex-face event causing the removal of the vertex.

Thus, each object pair may be tested using either a dynamic binary search tree or a static point location data structure, in logarithmic time per pair. □

4 General Polyhedra

4.1 Ambiguity

Defining the straight skeleton of a general polyhedron is inherently ambiguous, unlike the cases for convex and orthogonal polyhedra. The ambiguity stems from the fact that, whereas convex polyhedra are defined uniquely by the planes supporting their faces, nonconvex polyhedra are defined by *both* the supporting planes and a given topology, which is not necessarily unique. Thus, while being offset, a polyhedron can propagate from a given state into multiple equally valid topological configurations. (This issue was alluded to in [8].) A simple example is shown in Fig. 3(a). The problem is illustrated w.r.t. two boundary pieces—a wedge, A, and a tabletop, B—that are growing relative to each other. Due to the angle of the two front planes of A, the growing wedge eventually grows past the tabletop. The issue is to determine how the wavefronts continue growing. Possible choices include: (i) The wedge A grows through to the other side of B when A reaches the edge of B and moves past the edge; (ii) The wedge continues growing forward, but is blocked from growing downward by clipping it with the plane defined by the top of the tabletop; (iii) The wedge suddenly projects into the empty space in front of the table and continues growing out from there. In fact, all suggestions above cause a contradiction or a noncontinuous propagation of the wavefront. The actual solution that we chose is to blunt the front end of the wedge A by clipping it with the plane defined by the side of the tabletop.

A more general example of the ambiguity of the propagation of the skeleton is shown in Fig. 3(b). The figure shows a vertex of degree 5, and two possible topologies during the propagation. This is the so-called weighted-rooftop problem: Given a base polygon and slopes of walls, all sharing one vertex, determine the topology of the rooftop of the polygon, which does not always have a unique solution. In our definition of the skeleton, we define a consistent method for the initial topology and for establishing topological changes while processing the algorithm's events, based on the 2D weighted straight skeleton (see Section 4.3).

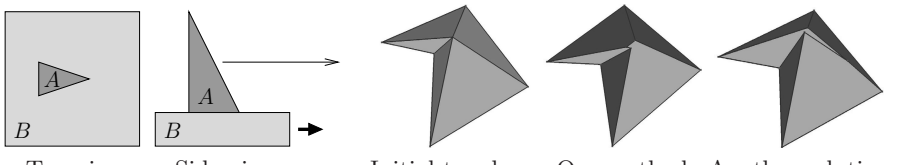


Fig. 3. 3D skeleton ambiguity

4.2 A Combinatorial Lower Bound

Theorem 6. *The complexity of a 3D skeleton for a simple polyhedron is* $\Omega(n^2\alpha^2(n))$ *in the worst case, where* $\alpha(n)$ *is the inverse of the Ackermann function.*

Proof. (Sketch) We use an example (see Fig. 4), in which a sequence of triangular prisms result in a growing wavefront whose complexity is that of the upper envelope of n line segments, that is, $\Omega(n\alpha(n))$ [22]. We attach two such sequences of prisms to the "floor" and "ceiling" of the polyhedron, obtaining two growing wavefronts which produce $\Omega(n^2\alpha^2(n))$ skeletal features. □

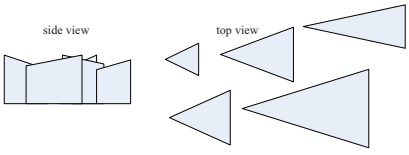


Fig. 4. Illustrating 3D skeleton complexity

4.3 The Algorithm

Our algorithm is an event-based simulation of the propagation of the boundary of the polyhedron. Events occur whenever four planes, supporting faces of the polyhedron, meet at one point. At these points the propagating boundary undergoes topological events. The algorithm consists of the following steps:

1. Collect all possible initial events.
2. While the event queue is not empty:
   (a) Retrieve the next event and check its validity. If not valid, go to Step 2.
   (b) Create a vertex at the location of the event and connect to it the vertices participating in the event.
   (c) Change the topology of the propagating polyhedron according to actions in Step 2(b). Set the location of the event to the newly-created vertices.
   (d) Create new events for newly-created vertices, edges, and faces and their neighbors, if needed.

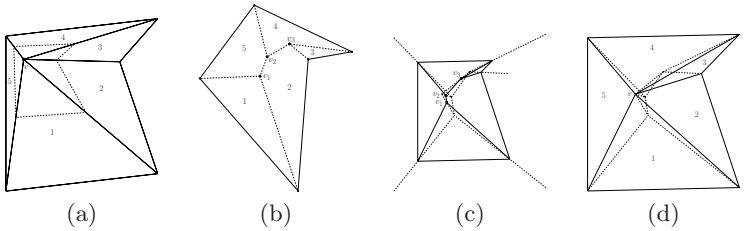


Fig. 5. Changing the initial topology of a vertex of degree ≥ 4 (skeleton in dashed lines): (a) The original polyhedron. Vertex v has degree 5; (b) The cross-section and its weighted straight skeleton. Vertex v becomes three new vertices v_1, v_2, v_3; (c) The straight skeleton of the polyhedron. Vertex v spawned three skeletal edges; (d) The propagated polyhedron. Vertices v_1, v_2, v_3 trace their skeletal edges.

We next describe the different events and how each type is dealt with. The procedure always terminates since the number of all possible events is bounded from above by the number of combinations of four propagating faces.

Initial Topology. At the start of the propagation, we need to split each vertex of degree greater than 3 into several vertices of degree 3 (see Fig. 5). This is the ambiguous situation discussed earlier; it can have several valid solutions. Our approach is based on cutting the faces surrounding the vertex with one or more planes (any cutting plane intersecting all faces and parallel to none suffices), and finding the weighted straight skeleton of the intersection of these faces with the cutting plane, with the weights determined by the dihedral angles of these faces with the cutting plane, after an infinitesimally-small propagation. The topology of this 2D straight skeleton tells us the connectivity to use subsequently, and always yields a unique valid solution. In the full version of the paper we detail the application of this method for all types of vertices.

Collecting Events. In the full version of the paper we describe how events are collected, classified as valid or invalid, and handled by the algorithm. In a nutshell, each event arises from interactions of features of the wavefront, and gives rise to potential future events. However, a potential event may be found invalid already when it is created, or later when it is fetched for processing. Each valid event results in the creation of features of the skeleton, and in a topological change in the structure of the propagating polyhedron.

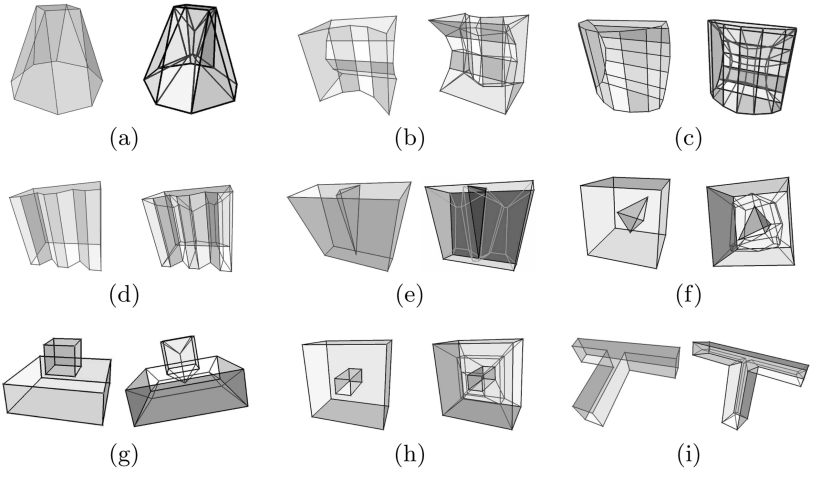
Handling Events. Propagating vertices are defined as the intersection of propagating planes. Such a vertex is uniquely defined by exactly three planes, which also define the three propagating edges adjacent to the vertex. (When an event creates a vertex of degree greater than 3, we handle it as as in the initial topology.) The topology of the polyhedron remains unchanged during the propagation between events. The possible events are:

1. **Edge Event.** An edge vanishes as its two endpoints meet, at the meeting point of the four planes around the edge.
2. **Hole Event.** A reflex vertex (adjacent to three reflex edges, called a "spike") runs into a face. The three planes adjacent to this vertex meet the plane of the face. After the event, the spike meets the face in a small triangle.
3. **Split Event.** A ridge vertex (adjacent to one or two reflex edges) runs into an opposite edge. The faces adjacent to the ridge meet the face adjacent to the twin of the split edge. This creates a vertex of degree greater than 3, handled as in the initial topology.
4. **Edge-Split event.** Two reflex edges cross each other. Every edge is adjacent to two planes.
5. **Vertex event.** Two ridges sharing a common reflex edge meet. This is a special case of the edge event, but it has different effects, and so it is considered a different event. Vertex events occur when a reflex edge runs twice into a face, and the two endpoints of this edge meet.

Data Structures. We use an event queue which holds all possible events sorted by time, and a set of propagating polyhedra, initialized to the input polyhedron, after the initialization of topology. The used structure is a generalization of the SLAV structure in 2D. We provide the details in the full version of the paper.

Running Time. Let n be the total complexity of the polyhedron, r be the number of reflex vertices (or edges), and k the number of events. For collecting the initial events, we iterate over all vertices, faces, and edges. Edge events require looking at each edge's neighborhood, which is done in $O(n)$ time. Finding hole events requires considering all pairs of a reflex vertex and a face. This takes $O(rn)$ time. Computing a split event is bounded within the edges of the common face, but this can take $O(rn)$ time, and computing edge-split events takes $O(r^2)$ time.

The algorithm computes and processes events. For a convex polyhedron, only edge events are created, each one computed locally in $O(1)$ time. However, for a general polyhedron, every edge might be split by any ridge and stabbed by any



Object	Object Vertices	Edges	Facets	Skeleton Vertices	Edges	Faces	Cells	Time (Sec.)
Objects								
(a)	12	20	10	8	24	25	10	0.312
(b)	20	30	12	25	60	46	12	0.719
(c)	28	42	16	45	104	74	16	0.567
(d)	20	30	12	16	42	37	12	0.188
(e)	20	18	9 (+1 hole)	15	45	56	9	0.250
(f)	12	18	10	21	48	37	10	0.484
(g)	16	24	11	6	21	25	11	0.177
(h)	16	24	11	12	36	33	11	0.146
(i)	16	24	10	12	32	29	10	0.172

(j) Statistics and running times

Fig. 6. Sample objects

spike. In addition, new spikes and ridges can be created when events are processed, and they have to be tested against all other features of their propagating component. Since $O(1)$ vertices and edges are created in every event, every event can take $O(n)$ time to handle. (The time needed to perform queue operations per a single event, $O(\log n)$, is negligible.) The total time needed for processing the events is, thus, $O(kn)$. This is also the total running time.

We have implemented the algorithm for computing the straight skeleton of a general polyhedron in Visual C++ .NET2005, and experimented with the software on a 3GHz Athlon 64 processor PC with 1GB of RAM. We used the CGAL library to perform basic geometric operations. The source code consists of about 6,500 lines of code. Fig. 6 shows the straight skeletons of a few simple objects, and the performance of our implementation.

References

1. Aichholzer, O., Aurenhammer, F.: Straight skeletons for general polygonal figures in the plane. In: Cai, J.-Y., Wong, C.K. (eds.) COCOON 1996. LNCS, vol. 1090, pp. 117–126. Springer, Heidelberg (1996)
2. Aichholzer, O., Aurenhammer, F., Alberts, D., Gärtner, B.: A novel type of skeleton for polygons. J. of Universal Computer Science 1(12), 752–761 (1995)
3. Barequet, G., Goodrich, M.T., Levi-Steiner, A., Steiner, D.: Contour interpolation by straight skeletons. Graphical Models 66(4), 245–260 (2004)
4. Bittar, E., Tsingos, N., Gascuel, M.-P.: Automatic reconstruction of unstructured 3D data: Combining a medial axis and implicit surfaces. Computer Graphics Forum 14(3), 457–468 (1995)
5. Blum, H.: A transformation for extracting new descriptors of shape. In: Wathen-Dunn, W. (ed.) Models for the Perception of Speech and Visual Form, pp. 362–380. MIT Press, Cambridge (1967)
6. Cheng, S.-W., Vigneron, A.: Motorcycle graphs and straight skeletons. In: Proc. 13th Ann. ACM-SIAM Symp. on Discrete Algorithms, pp. 156–165 (January 2002)
7. Culver, T., Keyser, J., Manocha, D.: Accurate computation of the medial axis of a polyhedron. In: Proc. 5th ACM Symp. on Solid Modeling and Applications, New York, NY, pp. 179–190 (1999)
8. Demaine, E.D., Demaine, M.L., Lindy, J.F., Souvaine, D.L.: Hinged dissection of polypolyhedra. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 205–217. Springer, Heidelberg (2005)
9. Demaine, E.D., Demaine, M.L., Lubiw, A.: Folding and cutting paper. In: Akiyama, J., Kano, M., Urabe, M. (eds.) JCDCG 1998. LNCS, vol. 1763, pp. 104–118. Springer, Heidelberg (2000)
10. Dey, T.K., Zhao, W.: Approximate medial axis as a Voronoi subcomplex. Computer-Aided Design 36, 195–202 (2004)
11. Eppstein, D.: Dynamic Euclidean minimum spanning trees and extrema of binary functions. Discrete & Computational Geometry 13, 111–122 (1995)
12. Eppstein, D.: Fast hierarchical clustering and other applications of dynamic closest pairs. ACM J. Experimental Algorithmics 5(1), 1–23 (2000)
13. Eppstein, D., Erickson, J.: Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. Discrete & Computational Geometry 22(4), 569–592 (1999)

14. Foskey, M., Lin, M.C., Manocha, D.: Efficient computation of a simplified medial axis. J. of Computing and Information Science in Engineering 3(4), 274–284 (2003)
15. Haunert, J.-H., Sester, M.: Using the straight skeleton for generalisation in a multiple representation environment. In: ICA Workshop on Generalisation and Multiple Representation (2004)
16. Held, M.: On computing Voronoi diagrams of convex polyhedra by means of wavefront propagation. In: Proc. 6th Canadian Conf. on Computational Geometry, pp. 128–133 (August 1994)
17. Price, M.A., Armstrong, C.G., Sabin, M.A.: Hexahedral mesh generation by medial surface subdivision: Part I. Solids with convex edges. Int. J. for Numerical Methods in Engineering 38(19), 3335–3359 (1995)
18. Sharir, M.: Almost tight upper bounds for lower envelopes in higher dimensions. Discrete & Computational Geometry 12, 327–345 (1994)
19. Sheehy, D.J., Armstrong, C.G., Robinson, D.J.: Shape description by medial surface construction. IEEE Trans. on Visualization and Computer Graphics 2(1), 62–72 (1996)
20. Sherbrooke, E.C., Patrikalakis, N.M., Brisson, E.: An algorithm for the medial axis transform of 3d polyhedral solids. IEEE Trans. on Visualization and Computer Graphics 2(1), 45–61 (1996)
21. Tănase, M., Veltkamp, R.C.: Polygon decomposition based on the straight line skeleton. In: Proc. 19th Ann. ACM Symp. on Computational Geometry, pp. 58–67 (June 2003)
22. Wiernik, A., Sharir, M.: Planar realizations of nonlinear Davenport-Schinzel sequences by segments. Discrete & Computational Geometry 3, 15–47 (1988)