Skeleton computation of orthogonal polyhedra

J. Martinez\textsuperscript{1}, M. Vigo\textsuperscript{1,2} and N. Pla-Garcia\textsuperscript{1,2}

\textsuperscript{1}Polytechnical University of Catalonia (UPC), Barcelona, Spain
\textsuperscript{2}Bioengineering Institute of Catalonia (IBEC). Barcelona, Spain

Abstract

Skeletons are powerful geometric abstractions that provide useful representations for a number of geometric operations. The straight skeleton has a lower combinatorial complexity compared with the medial axis. Moreover, while the medial axis of a polyhedron is composed of quadric surfaces the straight skeleton just consist of planar faces. Although there exist several methods to compute the straight skeleton of a polygon, the straight skeleton of polyhedra has been paid much less attention. We require to compute the skeleton of very large datasets storing orthogonal polyhedra. Furthermore, we need to treat geometric degeneracies that usually arise when dealing with orthogonal polyhedra. We present a new approach so as to robustly compute the straight skeleton of orthogonal polyhedra. We follow a geometric technique that works directly with the boundary of an orthogonal polyhedron. Our approach is output sensitive with respect to the number of vertices of the skeleton and solves geometric degeneracies. Unlike the existing straight skeleton algorithms that shrink the object boundary to obtain the skeleton, our algorithm relies on the plane sweep paradigm. The resulting skeleton is only composed of axis-aligned and 45° rotated planar faces and edges.

Categories and Subject Descriptors (according to ACM CCS): Computer graphics [I.3.5]: Computational Geometry and Object Modeling—

1. Introduction

One of the challenges of the BioCAD field is to understand the morphology of the pore space of bone, biomaterials, rocks, etc. There exist several approaches that obtain a network with pores and connections between them [VAGT08] or that characterize plate and rod elements [SM06], which rely on a previous computation of a surface skeleton.

In this work we follow a geometric approach to compute the skeleton of orthogonal polyhedra. We have chosen this approach because geometric methods perform fewer steps compared to thinning methods, as they do not need to work at voxel level. As real datasets obtained from $\mu$CT tend to be very large, computing their skeleton with thinning approaches is very time consuming. We require a robust and fast algorithm that extracts a skeletal representation with low combinatorial complexity.

We compute the Voronoi Diagram [Aur91] with the $L_\infty$ distance ($V_{D_{\infty}}$) of the input orthogonal polyhedron that coincides with the straight skeleton [AAAG95]. Our method relies on a sweep-line scheme applied to orthogonal polygons [MVPGA10] that we extend to three dimensions. Furthermore, our algorithm is capable of dealing with any kind of geometric degeneracies that commonly arise in orthogonal polyhedra.

2. Related work

The most well-known skeletal representation is the medial axis introduced by Blum [Blu67]. One way to define the medial axis is as the locus of centres of maximally inscribed balls. The medial axis is a subset of the Voronoi diagram under the Euclidean metric [Kir79]. The combinatorial complexity of the Voronoi diagram of a polyhedron is high. The best known upper bound on the combinatorial complexity is $O\left(n^{3+\varepsilon}\right)$ for any positive $\varepsilon$, where $n$ is the number of faces, edges and vertices of the polyhedron [SA95]. The medial axis of an object is homotopically equivalent to the object [Lie03]. The medial axis of a polyhedron is a piecewise algebraic surface composed of quadric surfaces.

Methods to compute the medial axis can be classified into three main families: thinning, distance field and geometric methods [CS07]. Most of them that apply to discretized data, e.g., images or volumes, are based on thinning and distance
field approaches. Thinning methods consist of iteratively removing points from the object boundary without changing its topology [LLS92]. Distance field approaches rely on the previous computation of the distance field [JBS06]. These methods are linear in the number of voxels. Geometric methods are generally applied to polygons and polyhedra and are commonly based on the Voronoi diagram.

Several geometric algorithms have been proposed to compute the medial axis of a polyhedron. In [Mi93] a calculation of all the points equidistant to four features allows to reconstruct the Voronoi diagram of polyhedra. Held [He94] propose a wavefront propagation algorithm only suitable for convex polyhedra. Sherbrooke et al. [SPP06] uses a tracing classification scheme. The vertices of the medial axis are connected by tracing the adjacent edges, and the faces of the medial axis are found by traversing closed loops of vertices and edges. A similar approach uses exact arithmetic and works using algebraic curves and surfaces [CKM04]. A divide and conquer approach is described in [SRX07]. As the computation proceeds a dual structure of the shape is broken up into pieces each representing a simpler part of the medial axis.

The medial axis is highly sensitive to small changes in shape boundary [ABE09]. Geometric algorithms for computing approximate representations of the medial axis try to reduce this instability to noise and high combinatorial complexity. Brandt and Algazi [BA91] showed that the Delaunay triangulation of a sufficiently dense set of samples contains a reconstruction of the boundary as a subset of its edges. Attali and Montavert [AM97] showed that a three dimensional shape can be approximated by a finite union of balls. Amenta and Kolluri [AK01] noticed that, given a sample of points on boundary shape, the union of a subset of the Voronoi balls approximates the original shape. The medial axis can also be approximated directly from the Voronoi diagram [DZ02]. The use of the Delaunay triangulation to obtain the skeleton was also explored [SAR95]. Etzion and Rappoport [ER99] construct the Voronoi diagram of a polyhedron by separating the computation of the symbolic and geometric parts of the Voronoi diagram. In some applications, an approximate skeleton with only one dimensional geometry is required [DS06]. Foskey et al. [FLM03] and Sud et al. [SFM05] compute a simplified medial axis that rely on the angle formed by the medial axis and its closest neighbours on the surface. Chazal and Lieutier [CL05] proposed the $\lambda$-medial axis, that contains the set of medial axis points whose closest neighbours on the boundary cannot be enclosed in a ball smaller than a global threshold parameter $\lambda$. Recently, Giesen et al. [GMPW09] introduced the scale axis transform that is based on the medial axis transform and the simplification of the shape under multiplicative scaling in order to capture the relevant features.

An alternative skeletal representation to the medial axis is the straight skeleton [AAAG95]. While the medial axis is defined using a distance function, the straight skeleton relies on a shrinking process in which the edges of the polygon are moved inwards, parallel to themselves at a constant speed. Each vertex moves along the angular bisection of its incident edges. The straight skeleton of a polygon is only composed of straight line segments, while the medial axis of a polygon may involve parabolic curves. Eppstein and Erickson [EE99] propose an algorithm to compute the straight skeleton that simulates a sequence of collisions between edges and vertices during the shrinking process, using a complex technique for maintaining extrema of binary functions [Epp98]. Recently Das et al. [DMN10] introduced a deterministic algorithm for computing the straight skeleton of monotone polygons in $O(n \log n)$.

The straight skeleton formulation can be extended to polyhedra. It can be defined in terms of an offset process in which the faces move inward at a constant speed. The sequence of collisions between features of the polyhedron define the skeleton. In three dimensions, the straight skeleton is composed of points, edges and planar faces. The combinatorial complexity of the straight skeleton of orthogonal polyhedra is $O\left(n^2\right)$, where $n$ is the number of vertices [BEGV08]. Note that the straight skeleton has a lower combinatorial complexity compared with the medial axis.

Definition 1 The $L_\infty$ distance between two points $(p,q)$, with coordinates $p_i$ and $q_i$, is $D_\infty(p,q) = \max(||p_i - q_i||)$. In three dimensions the $L_\infty$ distance corresponds to the side length of the smallest isothetic cube touching $p$ and $q$.

Definition 2 The Voronoi diagram of polyhedron $P$ under the $L_\infty$ metric $(VD_\infty)$ is a subdivision of the space into cells, such that the cell associated with a face $f \in P$ comprises the points in space for which $f$ is closer under the $L_\infty$ metric than all other faces of $P$. The $VD_\infty$ of a polyhedron in general position decomposes the space in a set of planar faces, edges and vertices $L_\infty$ equidistant to two, three and four features of the polyhedron respectively.

Property 1 The straight skeleton of an orthogonal polyhedron $P$ coincides with the $VD_\infty$ of the interior of $P$.

To the best of our knowledge the only algorithm that addresses the construction of straight skeletons of polyhedra was presented by Barequet et al. [BEGV08]. Their method relies on a shrinking of the object boundary and it is restricted to polyhedra in general position. In the case of orthogonal polyhedra, they compute for each pair of features the time at which the interaction would happen. The algorithm has $O\left(n^3 \log n\right)$ time complexity, where $n$ is the number of vertices of the polyhedron. A more complex output sensitive algorithm that uses orthogonal range searching techniques is outlined in their paper.
3. Algorithm overview

Sweep-line algorithms move a line across the plane, stopping at certain event points. Geometric operations are restricted to geometric objects that either intersect or are in the immediate vicinity of the sweep line whenever it stops, and the complete solution is available once the line has passed over all objects. A sweep-line algorithm used to extract the Voronoi diagram of a set of points under the Euclidean metric was introduced by Fortune [For87]. Papadopoulou and Lee [PL01] presented a sweep-line approach to compute the $VD_{\infty}$ of planar straight graphs that is strongly connected with our algorithm. We extend to three dimensions the sweep-line approach by considering a sweeping plane instead of a line. Our technique involves extracting the $VD_{\infty}$ of orthogonal polyhedra by a sweeping process.

Let $P$ be an orthogonal polyhedron with axis-aligned faces. Let $L$ be the sweep plane perpendicular to the $z$-axis ($z = t$). Our sweep process is done by decreasing the value of $t$.

**Definition 3** At any moment $t$ of the sweeping process, $L$ partitions the set of faces, edges and vertices of $P$ into three subsets: $A_t$ corresponds to those that lie fully above of $L$, $B_t$ corresponds to those that intersect $L$, and $C_t$ corresponds to those that lie fully below of $L$. $C_t$ is composed of the portion of $L$ that lies above $L$. $L_P$ is the set of faces of $P$ contained in $L$ plus the portion of $L$ that belongs to the interior of $P$.

The invariant is that at every instant $t$ of the sweeping process the $VD_{\infty}$ of $A_t \cup C_t \cup L$ is computed.

**Definition 4** A ray is a half-line that corresponds to a Voronoi edge $e$ of the $VD_{\infty}$, with its origin in a Voronoi vertex or a vertex of $P$. The end of the Voronoi edge $e$ remains to be calculated. We say that a ray is closed when the remaining vertex of $e$ is calculated. A ray is the bisector defined by three oriented faces of $P$. We consider that two rays are equivalent if they are defined by the same three faces.

**Property 2** Four axis-aligned faces define a cube if two of them share the same orthogonal direction, have opposed orientation, define a bisector, and the other two faces correspond to the two remaining orthogonal directions.

The sweep plane $L$ stops whenever a new event is found. The events are ordered according its priority. The priority of an event is given by the lexicographical order of its position. Two different types of events may occur during the sweep (see Figure 1):

- **Vertex event**: Corresponds to a vertex $v$ of $P$ and a ray associated to $v$. There can be more than one vertex event per vertex, because more than one ray may emanate from a vertex (see Section 4). The priority of a vertex event is defined by the position of the vertex.
- **Junction event**: Corresponds to the centre of a cube touching four faces of $P$ (see Property 2). A junction event is valid if it corresponds to a Voronoi vertex of $VD_{\infty}$. The priority of a junction event is given by the vertex of its cube that have the lowest lexicographic priority.

![Figure 1: Straight skeleton computation of a rectangular box.](image)

If the priority of a vertex event and a junction event coincide, the vertex event is processed first. If two junction events have the same priority, the junction event whose cube has the bigger side length is processed first. If two vertex events have the same priority or the input polyhedron is not in a general position this criterion will fail to sort the events. We propose a technique to treat geometric degeneracies based on a simulated perturbation that produces a valid sequence of events (see Section 9). Our approach has certain connection with the simulation of simplicity proposed by Edelsbrunner et al. [EM90].

**Definition 5** The wavefront is the portion of the $VD_{\infty}$, at an instant $t$, induced by $L_t$.

**Proposition 1** The wavefront of an orthogonal polyhedron is composed of a set of planar faces that give a single-valued linear piecewise function defined by the sweep plane $L$. Thus, the wavefront is monotone in the $z$-axis direction.

Proof Let $p = (x, y, z)$ be a point in $L_t$, in face $f$ of $L_t$, and let $p' = (x, y, z')$ be the point with maximal $z'$ value in the Voronoi cell of $f$. The maximal cube centered at $p'$ is empty, and contains every cube centered at a point on the segment $p, p'$, hence all of those are empty as well, and must belong to the Voronoi cell of $f$. So there can be no other point of wavefront between $p$ and $p'$.

An overview of our technique is shown in the Algorithm 1. The straight skeleton is progressively computed by processing a sequence of events that correspond to its vertices. Initially, $E$ is the ordered set of events induced by the vertices of the input object. We process the events of $E$ until it is empty. Every time we process a junction event we have to check if it is valid. Otherwise, we discard the junction event. The criterion used to check the validity of a junction event is discussed in the Section 5. A valid event may generate new events that will be inserted in $E$.

We maintain the set of rays $R$, that represent the set of Voronoi edges that remain to be closed. We also maintain...
The vertices of a two-manifold orthogonal polyhedron can be classified into three types depending on the number of incident edges and faces [Agu98] (see Figure 2). A vertex with three incident faces and edges is called V3. In order to simplify the algorithm we follow the approach proposed in [BEGV08], where the vertices with more than three incident faces are split into V3 vertices. A vertex with four incident edges and faces is called V4. A V4 vertex has two coplanar incident faces with opposite orientation. By combining those coplanar incident faces with the remaining two faces we obtain two distinct rays that induce two vertex events. Finally, a vertex with six incident faces and edges is called V6. A V6 vertex has three pairs of coplanar faces for each orthogonal direction. We arbitrarily select two coplanar faces and merge them into a single face. By combining the merged face and its two pairs of neighbouring faces we obtain two different rays that induce two vertex events.

5. Junction event

A junction event corresponds to the centre of the cube defined by four faces (see Property 2). A valid junction event coincides with a Voronoi vertex of $V D_{\infty}$. The priority of a junction event represents the point where the junction is valid. A junction event may be invalidated before we reach its priority point. Every valid junction event generates a set of new rays to be inserted in $R$, and each new ray can generate new junction events. We distinguish two types of junction events depending on the kind of ray junction:

- **Ray-ray junction**: Two rays of $R$ may define a Voronoi vertex. Two new rays emanate from the junction point (see Figure 3).
- **Ray-face junction**: A ray of $R$ and a face of $P$ may define a Voronoi vertex. Three new rays emanate from the junction point (see Figure 4).

5.1. Ray-ray junction events

A valid ray-ray junction event involves two rays that define a cube (see Property 2) whose centre is a Voronoi vertex, and that have two shared and two unshared faces between them (see Figure 3). We must check the validity of the junction. A ray-ray junction event is valid if the two rays exist in $R$. The two shared faces may represent a collapsed edge to zero length of the input polyhedron. If the ray-ray junction event is valid we generate two new rays by combining the pair of unshared faces with each shared face. We also add two edges in the skeleton graph between the origin of each ray and the junction vertex.

5.2. Ray-face junction events

A valid ray-face junction event corresponds to a Voronoi vertex defined by a ray and a face (see Figure 4). A ray-face junction event is valid if the ray exists in $R$. A ray-face junction event is not valid if any three of its four faces defining

---

**Algorithm 1 Algorithm overview**

```
R ← ∅
E ← All the vertex events induced by $P$
while $E \neq ∅$ do
  $e ← \text{pop}(E)$
  if $e$ is valid then
    Update the straight skeleton according to $e$
    Insert in $R$ new ray/s induced by $e$
    Remove from $R$ closed ray/s of $e$
    Update wavefront (see Section 6)
    Compute nearest junction of new ray/s (see Section 7) and insert in $E$ the new induced junction events
  end if
end while
```
a ray correspond to a equivalent ray erased from R. We only check for ray-face junctions if the ray contains any pair of faces bounding a reflex edge. If the ray-face junction is valid we generate three new rays by combining the three pairs of faces from the ray and the face. We also add an edge in the skeleton graph between the ray origin and the junction vertex.

Figure 3: Ray-ray junction. The rays shown in red define a cube whose centre is a Voronoi vertex. The edge \((f_1, f_2)\) collapses to zero length and two new rays shown in blue are created.

Figure 4: Ray-face junction. The ray shown in red defines a cube whose centre is a Voronoi vertex. The edge \((f_1, f_2)\) collapses to zero length and two new rays shown in blue are created.

6. Wavefront update

The wavefront is stored as a graph, where each vertex corresponds to a ray from \(R_0\) or to edges or vertices of \(F\). An edge in the wavefront connecting two rays indicates that both rays share two of their three inducing faces (see Figure 5). This leads to a maximum number of three possible neighbours per ray. Every time an event is processed, the wavefront is updated by removing from the graph the vertices associated to rays which have been closed and adding new rays created by the event. The position of a graph vertex is the centre of the cube defined by its associated ray and the sweep plane \(L\). By traversing the graph vertices that share a common face we obtain the associated Voronoi cell. We may insert or remove graph vertices and locally update the connectivity of the wavefront graph when a new event occurs. In the two dimensional case two bisectors, each one defined by a pair of edges, are neighbours in the wavefront if they share one edge. In this case, it suffices to classify the bisectors by its coordinate in a single orthogonal direction in order to know its upper and lower neighbour bisector. However, in three dimensions a ray may have more than two neighbours and it is not sufficient to classify them only by using its origin on a single orthogonal direction.

**Lemma 1** The wavefront update of a ray-ray junction can be done in \(O(1)\) time.

**Proof** After processing an event we have a set of new rays and a set of closed rays. There are at most three new rays and three closed rays. The graph vertices and edges associated to the set of closed rays are removed from the wavefront graph. The wavefront graph connectivity of the new rays inserted in \(R\) must be updated. We retrieve the set of neighbouring rays of all the rays that participated in the new event and try to connect them with the new rays. We connect two rays if they share two faces. There are at most six rays associated to the event that have at most eighteen neighbouring rays. □

**Lemma 2** The wavefront update of a ray-face junction can be done in \(O(\log n)\) time.

**Proof** If we process a ray-face junction event some wavefront graph edges may be divided by the newly created rays. Let a new ray be represented by the faces \((f_a, f_b, f_c)\) with origin in the junction event defined by the faces \(F = (f_a, f_b, f_c, f_d)\). If the pair of faces \((f_i, f_b)\), with \(i \in \{a, b, c\}\), corresponds to a reflex edge or the pair of faces \(F \setminus (f_i, f_b)\), correspond to an existing edge graph of the wavefront, then the new ray may split the edge graph. We retrieve the wavefront edge \(F \setminus (f_i, f_b)\) that contains the origin of the ray and split it. This update can be done in \(O(\log n)\) by storing the wavefront in a spatial data structure as proposed in Section 7.

Consider the Figure 5 where the ray \((f_1, f_3, f_6)\) will define a junction with the face \(f_2\) as the sweep plane descend along the z-axis. In the case of the new rays \((f_1, f_3, f_2), (f_5, f_6, f_2), (f_1, f_6, f_2)\) that will be created from the junction, the pair of faces \((f_5, f_6)\) correspond to a reflex edge and the pair of faces \((f_1, f_2)\) correspond to a graph edge that must be split. □

7. Computation of the nearest junction event of a ray

A fundamental step of our algorithm is the computation of the nearest junction event of a ray. Given the current set of rays \(R\) we have to retrieve the nearest junction event of a ray and insert it in \(E\).

If a ray already exists in \(R\) we remove both rays, defined
by the same faces and with opposed orientation, and add to the skeleton graph the Voronoi edge bounded by their origins. Otherwise, we have to compute its nearest junction and insert it in $E$. The total number of feasible ray-ray junctions of a ray is bounded by the number of neighbouring wavefront rays. There are at most three wavefront ray neighbours, so we have to check at the most three possible ray-ray junctions. Consider the example of the Figure 5. To retrieve the possible ray-ray junctions of the ray defined by the faces $(f_1, f_2, f_3)$ we just check for its neighbouring vertices in the wavefront graph $(f_1, f_3, f_4)$ and $(f_1, f_2, f_5)$.

The most time consuming step of the nearest ray junction computation is the detection of ray-face junctions. We search for this kind of junctions only if the ray is defined by any pair of faces bounding a reflex edge. First we consider a ray that have a negative orientation in the z-axis direction. We have to check the subset of edges of the wavefront graph that are neighbouring to the three ray faces. This problem can be reduced to two dimensions, because there is one common face between the ray and the neighbouring edges. Let a ray be defined by the faces $(f_a, f_b, f_c)$ and $(f_a, f_d)$ be an edge adjacent to the region corresponding to $f_a$ in the wavefront graph. We can check in a greedy form if the bisector bounded by the faces $(f_a, f_b)$ intersects any edge $f_i$. However, we are able to retrieve the intersection in $O(\log n)$ by using a sweep-line algorithm to compute the straight skeleton of orthogonal polygons as proposed by Martinez et al. [MVPGA10]. We will explain this step more fully and establish the $O(\log n)$ query time in a longer journal version of this paper. Consider the Figure 5 where the ray that has a negative orientation in the z-axis $(f_1, f_3, f_6)$ also has a reflex edge bounded by the faces $(f_3, f_6)$. We may have to check in a greedy form for ray-face junction with the neighbouring graph edges of faces $f_3$ and $f_5$. This problem is reduced to two dimensions by checking if the bisector of two edges $(f_3, f_6)$ defines a Voronoi vertex with the segments $f_3$ and $f_2$ contained in the face $f_1$. In case a ray has a positive orientation in the z-axis direction, we have to obtain the face of the wavefront that is being intersected. A greedy way is to check the ray intersection for all the wavefront faces. However, this problem is analogous to planar point location. Thanks to the Property 1 it is possible to project the wavefront into a two dimensional space.

An additional check must be done in case we have faces with holes in order to detect possible ray-face junction events. If we have holes in a face the wavefront connectivity between vertex rays is lost. Let $f_b$ be face with holes. When a new Voronoi face defined by the bisector faces $f_a$ and $f_b$ is created we obtain the set of rays belonging to any hole of $f_b$, that intersect the bisector between $f_a$ and $f_b$, and update if necessary their nearest junction event. We will show in longer journal version of this paper that this query can be done in $O(\log n)$ by using planar point location data structure.

8. Computational complexity

We have to process $k$ events that correspond to the number of vertices of the straight skeleton.

Lemma 3 The straight skeleton of orthogonal polyhedra can be straightforwardly computed in $O(kn)$ time.

Proof The ray-ray junction detection is done in constant time. The ray-face junction detection of a ray that has a negative orientation in the z-axis is done in $O(n)$ by checking all the neighbouring wavefront faces of the ray. The ray-face junction detection of a ray that has a positive orientation in the z-axis can be done in $O(n)$ using a greedy search along all the wavefront faces.

Lemma 4 The straight skeleton of orthogonal polyhedra can be computed in $O(k\log^c n)$.

Proof The ray-face junction detection of a ray that has a negative orientation in the z-axis is done in $O(\log n)$ by reducing the problem to two dimensions and using a sweep-line algorithm. By storing the wavefront in a planar point location data structure we can accelerate the junction detection of a ray that has a positive orientation in the z-axis, yielding $O(\log n)$ time per ray and $O(\log^c n)$ time per update of the wavefront, for a constant $c$ [ABG06].

9. Handling geometric degeneracies

Geometrical degeneracies commonly arise in case of orthogonal polyhedra. A robust algorithm must consider them. We provide a technique to treat degeneracies that relies on simple geometric comparisons. Unlike other approaches we do not perturb directly the input polyhedron in order to eliminate degeneracies. The computation of the straight skeleton is always based on its original coordinates.

The number of degenerate cases that have to be distinguished in the computation of the straight skeleton of polyhedra is high. However, in two dimensions a deterministic

Figure 5: Wavefront graph. On the left a 3D view of an object. On the right the status of its wavefront graph shown in red. The rays inducing the wavefront are shown in blue.
criterion is enough to overcome all the cases [MVPGA10]. When we detect a geometrical degeneracy we simulate a perturbation that removes the degeneracy. For simplicity, we will consider that we displace every face of the polyhedron by an infinitesimal amount. However, our approach only requires to make comparisons between the displacement values assigned to each face and does not directly rely on the coordinates of the polyhedron. We assume that it is possible to displace by an infinitesimal amount each face of an orthogonal polyhedron such that the polyhedron is still valid.

We distinguish between two kind of degeneracies: the induced by coplanar elements and by coincident geometric elements (see Figure 6). If a coplanar or coincident degeneracy is detected, the appropriate simulated perturbation is applied. The simulation is done by defining a total order between degenerate events that have the same priority and by avoiding unfeasible ray junctions when a degeneracy is detected. We assign two offsets to every face of the polyhedron. Those offsets represent a symbolical infinitesimal displacement of the face and help to break coplanar and coincident degeneracies respectively.

Proposition 2 It is possible to perturb a set of coplanar faces such that the degeneracy is removed.

Proof Let a coplanar face belong to a set \( F \) of coplanar faces. By enumerating every face of \( F \) we define its offset. This implies that \( F \) is totally ordered. We do not use the offset of coplanar faces to directly perturb them. This is because we may have a large collection of coplanar faces. A subset of faces of \( F \) that is going to be perturbed is still totally ordered. The new ordering of the subset defines the offset values used for the perturbation. The coplanar faces associated to a vertex \( V_6 \) require an additional restriction. When we merge an arbitrary selected pair of coplanar faces meeting at the \( V_6 \) we have to ensure that the other four coplanar faces have an offset such the perturbed result is still valid. One pair of faces defines a reflex edge and the other pair defines a convex one after the merging. We have to ensure that the offset of faces bounding the reflex edge is less than the offset of faces bounding the convex edge. Otherwise, the faces may intersect between them and lead to an incorrect result. Note that the set \( F \) still remains totally ordered after introducing these restrictions.

A singular case of coplanar degeneracy may be caused by the hole of a face. Imagine the hole of a face being extruded outwards. When two rays emanating from the face hole collide a coplanar degeneracy is induced by the entire hole face. We erase the emanating rays from this junction and add the bisector induced by the face hole in a post-processing step (see Section 9.4).

9.2. Coincident elements

Definition 7 Let \( \alpha \) be a constant positive value. A set of faces \( F \in P \) are coincident if \( |F| > 4 \) and if for every pair of faces \( f_1, f_2 \in F \), that have the same orthogonal direction, opposed orientation and define a bisector, we have \( D_\infty (f_1, f_2) = \alpha \).

Proposition 3 It is possible to perturb a set of coincident faces such that the degeneracy is removed.

Proof It is enough to define an offset based on the orthogonal direction of each face in order to break the degeneracy. We assign a different prime number for every orthogonal direction of each face in order to break the degeneracy. By enumerating every face of \( F \) we define its offset. This implies that \( F \) is totally ordered. We do not use the offset of coplanar faces to directly perturb them. This is because we may have a large collection of coplanar faces. A subset of faces of \( F \) that is going to be perturbed is still totally ordered. The new ordering of the subset defines the offset values used for the perturbation. The coplanar faces associated to a vertex \( V_6 \) require an additional restriction. When we merge an arbitrary selected pair of coplanar faces meeting at the \( V_6 \) we have to ensure that the other four coplanar faces have an offset such the perturbed result is still valid. One pair of faces defines a reflex edge and the other pair defines a convex one after the merging. We have to ensure that the offset of faces bounding the reflex edge is less than the offset of faces bounding the convex edge. Otherwise, the faces may intersect between them and lead to an incorrect result. Note that the set \( F \) still remains totally ordered after introducing these restrictions.

9.3. Simulating the perturbation

A pair of events in \( E \) may have the same priority if they are degenerated. Note that two vertex events associated to a \( V_4 \) or \( V_6 \) vertex are considered to be degenerate after we break them into two \( V_3 \) vertices. We need to decide which event is processed before the other one. We introduce an additional
comparison to avoid the ordering ambiguity. If two events have the same priority in an orthogonal direction we further check if they still have the same priority after the simulated perturbation.

When we compute the intersection of a ray with a face (see Section 7) it may intersect more than one face. We have to decide if the intersection is feasible according to the off-sets. We proceed in a similar manner as when ordering degenerate events. Let $F$ be the set of coincident faces at the intersection point. If two or more faces of $F$ are coplanar we have to check for coplanar intersection perturbation. If the degeneracy is still unsolved we have to check for coincident intersection perturbation. The intersection perturbation consists in moving the origin of the ray and the intersected face conforming to their offsets. We have to check if the ray still intersects the face after the perturbation. If we intersect an edge or vertex, we will displace it according to the offset and check if the ray still intersects the face.

9.4. Repairing coplanar ambiguities

We select a portion of the boundary of the volume equidistant to a set of coplanar faces. This arbitrary selection causes that the computed straight skeleton may not be a unique solution, indeed it is correct in the sense of the $L_\infty$ metric. The selected part of the boundary depends on the offsets assigned to resolve coplanar degeneracies. We select the central area of the volume in a post-processing step in order to obtain a unique solution (see Figure 7).

Figure 7: Reparation of a coplanar ambiguity induced by two coplanar faces. Straight skeleton is shown in red. The blue arrow is the central bisector induced by $f_1$ and $f_2$.

**Lemma 5** The straight skeleton coplanar ambiguities can be repaired in $O(k)$ time, where $k$ is the number vertices of the skeleton.

**Proof** We repair the vertices associated to two or more coplanar faces, where an arbitrary selection of the volume boundary has been done and locally repair the graph by selecting the central area. A set of skeleton graph vertices may induce an ambiguity if all the vertices coincide in a point, are connected between them, and contain more than one coplanar face. Let $F_c$ be the set of faces associated to all the coincident vertices and $F_c \subset F$: a subset of coplanar faces. We traverse the neighbouring edges that contain at least two faces of $F_c \setminus F$ until we reach an edge that contains the projection of the central bisector bounded by $F_c$. We also have to repair the coplanar degeneracy that may be implicitly generated by a hole of a face. For every vertex of the skeleton graph that belongs to the intersection of a rays emanating from the hole we consider a ray emanating from it in the same orientation of the face containing the hole. We have to retrieve the intersection of this new ray with the unconnected skeleton generated by the face containing the hole (see Figure 8).

Figure 8: Reparation of a coplanar ambiguity induced by a hole. Straight skeleton graph is shown in red. The blue arrows correspond to the central bisector induced by the hole.

10. Results

We computed the straight skeleton of some representative objects in order to check the algorithm robustness. We implemented the algorithm with $O(kn)$ time complexity (see Section 8). The algorithm has been implemented in C++ and the source code consists of about 3000 lines of code. Table 1 shows the results on a PC Intel E6600 2.40 Ghz with 3.2 GB RAM for seven sample objects that are shown in Figure 9. We previously remove the V or Voronoi edges that emanate from the vertices of $P$. We also have compared the presented approach with an optimized thinning approach [AVV07]. Our technique obtained better results.

<table>
<thead>
<tr>
<th>Object</th>
<th>Skeleton</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>F</td>
<td>V</td>
</tr>
<tr>
<td>(a)</td>
<td>24</td>
<td>14</td>
</tr>
<tr>
<td>(b)</td>
<td>24</td>
<td>14</td>
</tr>
<tr>
<td>(c)</td>
<td>20</td>
<td>13</td>
</tr>
<tr>
<td>(d)</td>
<td>32</td>
<td>30</td>
</tr>
<tr>
<td>(e)</td>
<td>981</td>
<td>571</td>
</tr>
<tr>
<td>(f)</td>
<td>4617</td>
<td>2743</td>
</tr>
<tr>
<td>(g)</td>
<td>3399</td>
<td>2016</td>
</tr>
</tbody>
</table>

Table 1: Values shown are number of faces F and vertices V of the object and its straight skeleton. Running time of the skeleton computation in seconds.

11. Conclusions and future work

We have presented a new algorithm to compute the straight skeleton of an orthogonal polyhedron. Although the medial
axis of polyhedra has been extensively explored, the straight skeleton of polyhedra has devised less interest. However, the combinatorial complexity of the straight skeleton is much lower compared with the medial axis and is just composed of planar faces. We proposed a geometric approach that works directly with the boundary of the object. As a consequence, the time complexity of our algorithm is constrained by the size of its boundary. On the contrary, the time complexity of thinning and distance field approaches is bounded by the volume of the discretized object. Unlike the existing straight skeleton approaches that simulate a shrink of the object boundary, we have used a plane-sweep algorithm. Furthermore, our algorithm is output sensitive with respect to the number of features of the output skeleton.

As our algorithm is designed for practical purposes, we were concerned about its robustness. We developed a technique to treat geometric degeneracies based on a simulated perturbation of the input polyhedron. Moreover, we solved the ambiguity induced by coplanar elements, which are common in orthogonal polyhedra. As an immediate result of our work, we want to compute the skeleton of large datasets in order to show the performance of our method. In addition, our approach may be extended to handle non-manifold orthogonal polyhedra that arise in voxel datasets.

The straight skeleton of orthogonal polyhedra skeleton is still highly sensitive to shape noise and it is not comparable with prior approximate approaches (see Section 2). For this reason, an interesting avenue for future work would be to develop and approximate approach starting from the formulation of the straight skeleton. Furthermore, we may be able to approximate the skeleton of non-orthogonal polyhedra by approximating it with orthogonal polyhedra.

12. Acknowledgements

This work has been partially supported by the project TIN2008-02903 of the Spanish government and by the IBEC (Bioengineering Institute of Catalonia). We would like to thank the anonymous reviewers for their valuable suggestions.

References


Figure 9: Sample objects. Input orthogonal polyhedron shown in gray. Straight skeleton shown in red. (a) Two coincident V4 vertices. (b) Repairing of a coplanar degeneracy induced by two pairs of coplanar faces. (c) Repairing of coplanar degeneracy induced by an extruded hole. (d) Eight coincident V6 vertices. (e) Hand. (f) Binzilla. (g) Camel and head detail. Note that only the Voronoi edges are drawn in objects (e), (f) and (g).