

A CGAL implementation of the Straight Skeleton of a Simple 2D Polygon with Holes

Fernando Cacciola
SciSoft
fernando_cacciola@hotmail.com

This short paper overviews the CGAL implementation of the Felkel's algorithm [Fe98] for the construction of a Straight Skeleton on the interior of a 2D Polygon with Holes.

The Straight Skeleton: Short Overview.

A Straight Skeleton is a unique partition of the plane induced by a set of planar straight-line figures. It was first introduced by Aichholzer et. al in 1995 for the interior of a polygon [Ai95], and generalized later for general (straight-line) figures in the plane [Ai98].

Consider the continuous inner offsetting of a simple (non self-intersecting) polygon. In such a *wavefront* or *grassfire* transformation, edges either contract or expand w.r.t. a vertex depending on the internal angle formed at it. At non-reflex internal angles ($<PI$), edges contract, while at reflex internal angles ($>PI$) edges expand. (In the degenerate case of a pseudo-vertex of angle PI the edge neither contracts nor expands). Under this transformation vertices move along the angular bisector of the *lines* subtending the edges. The topology of each subsequent offset polygon remains unaffected unless one of two possible things occur: (a) The edges E_{i-1} and E_{i+1} — adjacent to an edge E_i — collide and E_i *vanishes*, (b) An edge E_i collides simultaneously with two consecutive edges E_j and E_{j+1} which are expanding, splitting E_i at the intersection point. These two possible *events* changes the topology of next offset: after an Edge Event, the polygon has less edges; and after a Split Event, it is split in two different polygons.

The Straight Skeleton (SK) is a tree-like topological and geometric structure which synthesizes the wavefront described above.

The traces of the moving vertices form the bisectors of the SK, and the *instants* or points where *events* occur form the SK nodes. Skeleton nodes connect the bisectors (three or more). Each bisector, because it is given by the trace of a moving vertex, is defined by two edges and it's a segment of the angular bisector between the lines subtending the edges. Contour bisectors are those defined by edges which are consecutive in the source polygon while Inner bisectors are those defined by edges which are consecutive not in the source but in the subsequent offset polygons. Because in the wavefront transformation all edges move at the same speed along their respective perpendicular directions, events—which are the result of the collision between edges—occur at an instant which is uniquely given by the offset distance of the colliding moved edges to the lines subtending their original versions. Therefore, events are defined by the simultaneous collision of three or more offset edges, either contracted or expanded. Since edges contract or expand, the offset distance that corresponds to each event is the Euclidean distance between the *lines* subtending the edges and not between the edge's segments. Such an offset distance, or instant, uniquely determines the order of the events: events which correspond at a shorter distance (or a prior instant) occur first. It is easy to see that since any event changes the topology of the moving polygons, events can not be entirely foreseen ahead of time since the occurrence of an event directly affects the feasibility of foreseen events. At least, any foreseen event might be no longer applicable. Furthermore, the correctness of the algorithm primarily depends on the correct ordering of events.

The SK is topologically and *visually* similar to a Voronoi Diagram (VD). It partitions the polygon in faces, each of which is uniquely associated with a polygon edge. Unlike a VD, non-terminal reflex vertices do not have their own faces: all faces corresponds to edges (or terminal vertices for

the generalized case of an open figure). It is not a VD-like structure because it is not defined by a function of the distance between edges (though it is *dependent* on the distance between the *lines* subtending the edges).

Since the very definition of a Straight Skeleton is based on the continuous wavefront or grassfire propagation of the edges, it is specially suited for polygon offsetting. In particular, it can be used to obtain the so-called “mitered” offsetting where corners remain as such in the offset polygon (this is unlike the “rounded” offsetting produced using a Voronoi Diagram).

If the polygon is convex, the SK is exactly the same as the VD. Otherwise, reflex vertices have the effect of pushing the inner bisectors away from the vertex, shifting the “axis” from the center. Therefore, it is not suited for Medial Axis applications where “medial” is required to mean equidistant to the boundary. However, some Medial Axis (MA) applications use the MA as a shape descriptor. In this case, the SK offers the exact same synthetic power as a VD: A polygon can be reconstructed from a SK, thus a lower resolution polygon can be reconstructed from a pruned SK.

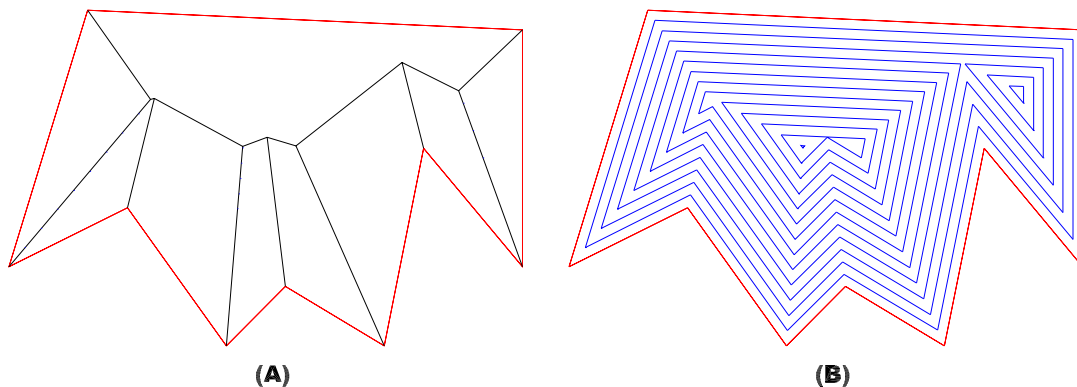


Figure 1: (a) Straight Skeleton (b) Mitered Offset polygons based on the SK

The representation of the Straight Skeleton in CGAL.

Since the SK is a partition of the polygon in faces associated with edges, it is natural to use a CGAL Halfedge Data Structure as the basis for the SK representation. Thus, the SK itself is given as a specialized HDS (a derived class).

Source polygon vertices and SK nodes are both represented as Vertices of the HDS.

Thus, A SK::Vertex holds: a point, properties which indicates if it is a non-convex/convex border vertex or an inner node, links to the incoming/outgoing edges and a link to the outgoing bisector.

Source polygon edges and bisectors are both represented as Halfedges of the HDS.

A halfedge is actually a *handle* wrapping the *body* which implements either an edge or a bisector. The handle itself provides the interface to query the common properties of both edges and bisectors: the segment/line defining the geometry of the edge/bisector, a property telling if it is a polygon edge or a bisector, and a link to the defining polygon edge (if this is a polygon edge already the link points to itself)

Just like any HDS, the SK user can traverse any face by following the halfedges. Additionally, the SK tree can be traversed also by following halfedges and faces.

The Implementation of the construction algorithm in CGAL

The construction algorithm implemented is essentially the same published by Felkel [Fe98], with the addition of Vertex Events as described in [Ep99] (the details of the algorithm is out of the scope of this presentation)

Roughly, the algorithm can be sketched as follows:

1. Initialization
 - Compute initial angular contour bisectors.
 - Compute initial Events, placing them in a priority queue ordered by their *instants*
 - The Split Events for all reflex contour bisectors.
 - The initial set of Edge Events for all consecutive intersecting contour bisectors.
2. Propagation
 - Process each Event in turn. This processing generates new bisectors which in turn produce new EdgeEvents.

As you can see from the sketch, the algorithm requires special predicates (for instance, to order the instants of the events) and constructions (for bisectors and events).

As with most geometric algorithms, its correctness can be made a exclusive function of its predicates if these are chosen carefully. In our case, the fundamental predicates are the relative ordering of events. It suffices that these predicates report correct answers to guarantee the correctness of the algorithm, even if the input polygon is not in general position (because these predicates are not convexity tests).

These predicates operate on polygon edges, bisectors and bisector intersections, so the algorithm requires special constructions that cooperate with the predicates. Therefore, the predicates and predicate-related-constructions needed by the algorithm are factored out from the algorithm itself and encapsulated on a traits class:

In order to allow the algorithm correctness to depend solely on the predicate correctness, the traits abstracts away the predicates input by means of special objects which are constructed by the traits class itself. That is, the predicates do not operate directly on intersection points or distances but on objects which encapsulate these.

The following are the objects constructed by the traits which participate in the predicates:

Bisectors: The trait is responsible for defining and constructing the geometric aspect of a bisector. (The topology of a bisector, i.e., its combinatorial structure, is governed by the algorithm)

Intersection positions: An Edge Event occurs when two non-consecutive edges collide simultaneously with some *non-opposite* edge. An Split Event occurs when two consecutive edges collide simultaneously with some *opposite* edge. These can be *seen* as the intersection of two bisectors, but is actually the simultaneous intersection of 3 *offset* edges. A performance-targeted traits can actually compute and store the intersection point, but a robustness-targeted implementation can simply cache a reference to the source edges involved in the simultaneous intersection and defer computations to the predicates operating on positions.

Event-instants: The ordering of events is given by the offset distance between the polygon edges and the Positions where the event occurs. A performance-targeted implementation can compute and store this distance, but a robustness-targeted implementation can simply cache the source edges involved in the simultaneous intersection that produces the event and defer computations to the predicates.

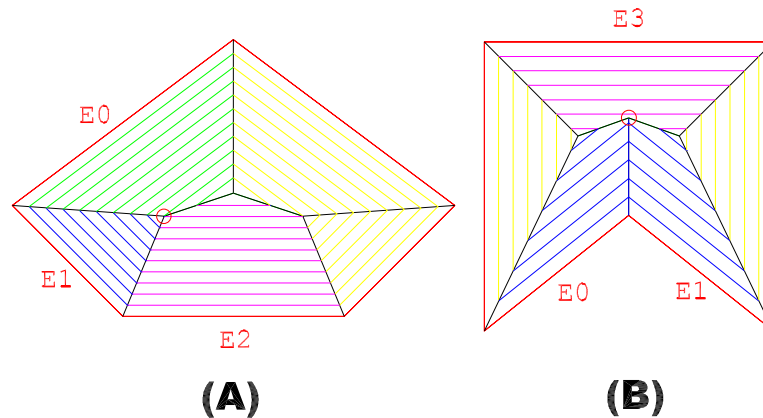


Figure 2

- (a) Edge Event: Non-consecutive edges E0 and E2 collide and E1 collapses.
 (b) Split Event: Consecutive Edges E0 and E1 collide simultaneously with opposite edge E3, which gets split.

The predicates offered by the traits operate on the objects mentioned above. Among other things, they order Positions (globally and w.r.t. a given bisector) and Event Instants, determine whether a Position is inside an offset region (the cone of influence of a contracting/expanding edge), determine whether Positions or Bisector are coincident, etc...

The default traits class which is currently implemented has been designed for fast unfiltered built-in floating-point types. Therefore, it actually computes bisector lines, intersection points and Euclidean distances. In order to compute bisector lines, it requires angle manipulation; and in order to test whether two positions are coincident, it requires a tolerance-based point equality test. Consequently, the default traits is parameterized on a Kernel Extension which adds support for a polar type and angle-based computations plus a user-defined point equality comparator.

Note, however, that the Kernel Extension provided is needed only for the default Traits, which is designed to trade robustness for speed. Unfortunately, such a speed-oriented traits is bound to fail in degenerate and near degenerate cases. Degenerancies in this context comes from a higher (>3) multiplicity of collisions or aparent collisions. For example, cocircular or colinear vertices induce events at the same instant, but due to numerical issues these theoretically equivalent instant will be computed with slightly different values and the ordering of the events will be affected, possibly compromising the correctness of the result. Note however that colinear or cocircular vertices do not represent a degenerate case in the RealRAM (with exact arithmetic).

Alternative traits implementations are possible and have been explored but were not actually implemented: Angle-base computations can be avoided if the Bisectors are represented without explicit geometry. Since the geometric definition of a bisector (and associated geometric operations) are defined by the Traits, this alternative characterization is possible without affecting the algorithm. Such a Traits require the intersection computations to operate on the edges directly, solving a system of linear equations. Similarly, Positions and Instants can be represented without any explicit value, deferring computations to the *Ordering* of those, which can be solved using only the source edges by solving a system of linear equations.

References

- [Ai95] O. Aichholzer. et. al. "A Novel Type of Skeleton for Polygons", 1995.
 [Ai98] O. Aichholzer. et. al. "Straight Skeletons for General Polygonal Figures in the Plane",
 [Fe98] P. Felkel. et. al. "Straight Skeleton Implementation", 1998.
 [Ep99] D. Eppstein. "Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions"