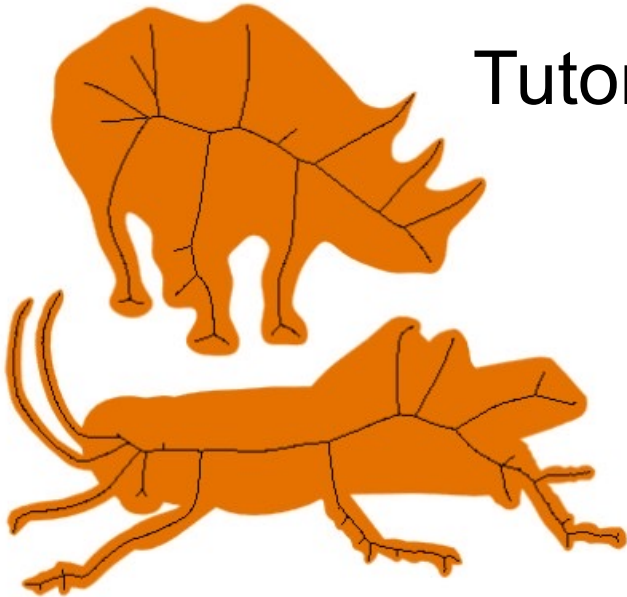
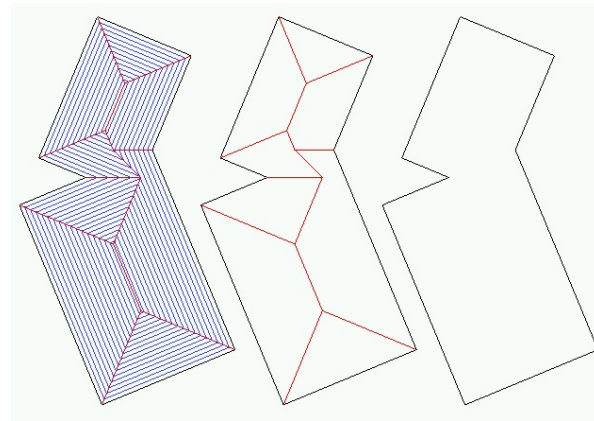
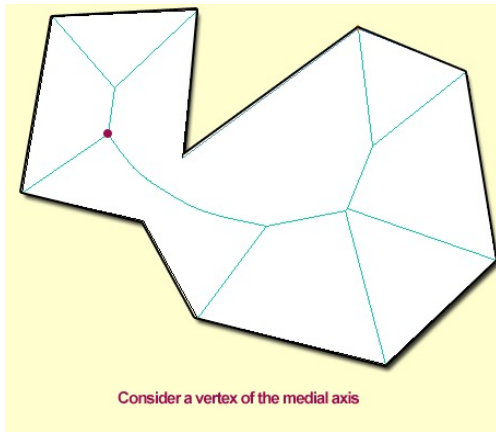

Polygonal Skeletons

Tutorial 3 – Computational Geometry



The Skeleton of a Simple Polygon

- A **polygon** is a closed contour in the plane, which might contain holes (which are simple polygons as well).
- A **skeleton** of a polygon is a partition of the polygon into regions, creating internal vertices, edges and faces.
- We will deal with two main types of skeletons: **The Medial Axis** and **The Straight skeleton**.

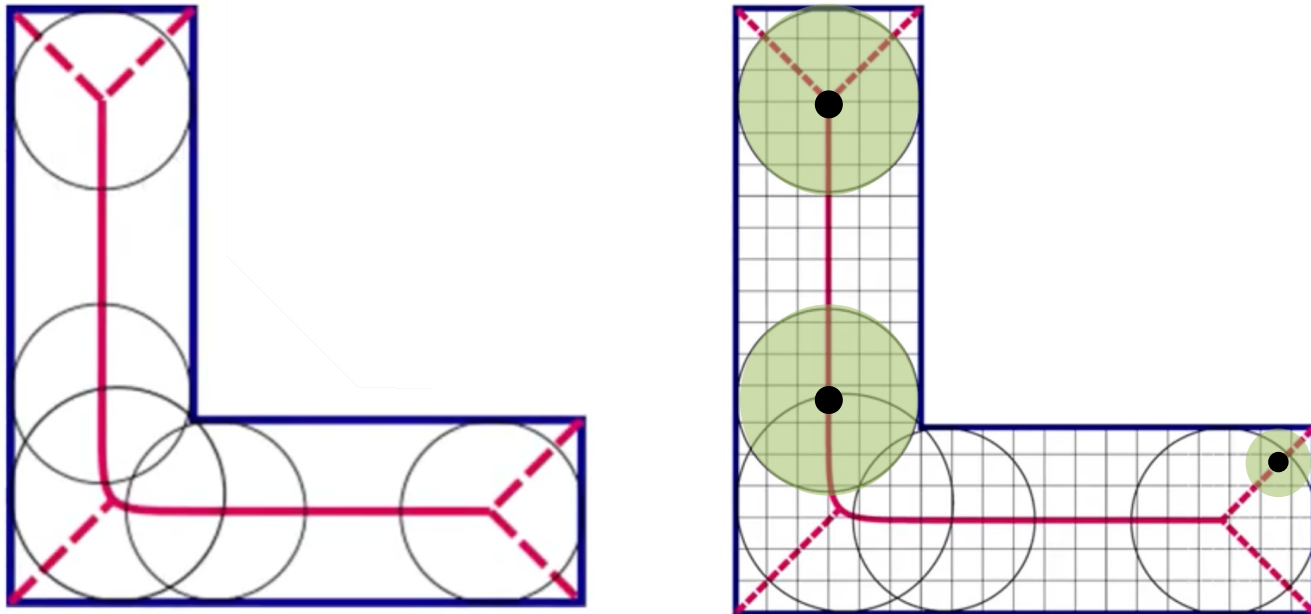


The Medial Axis

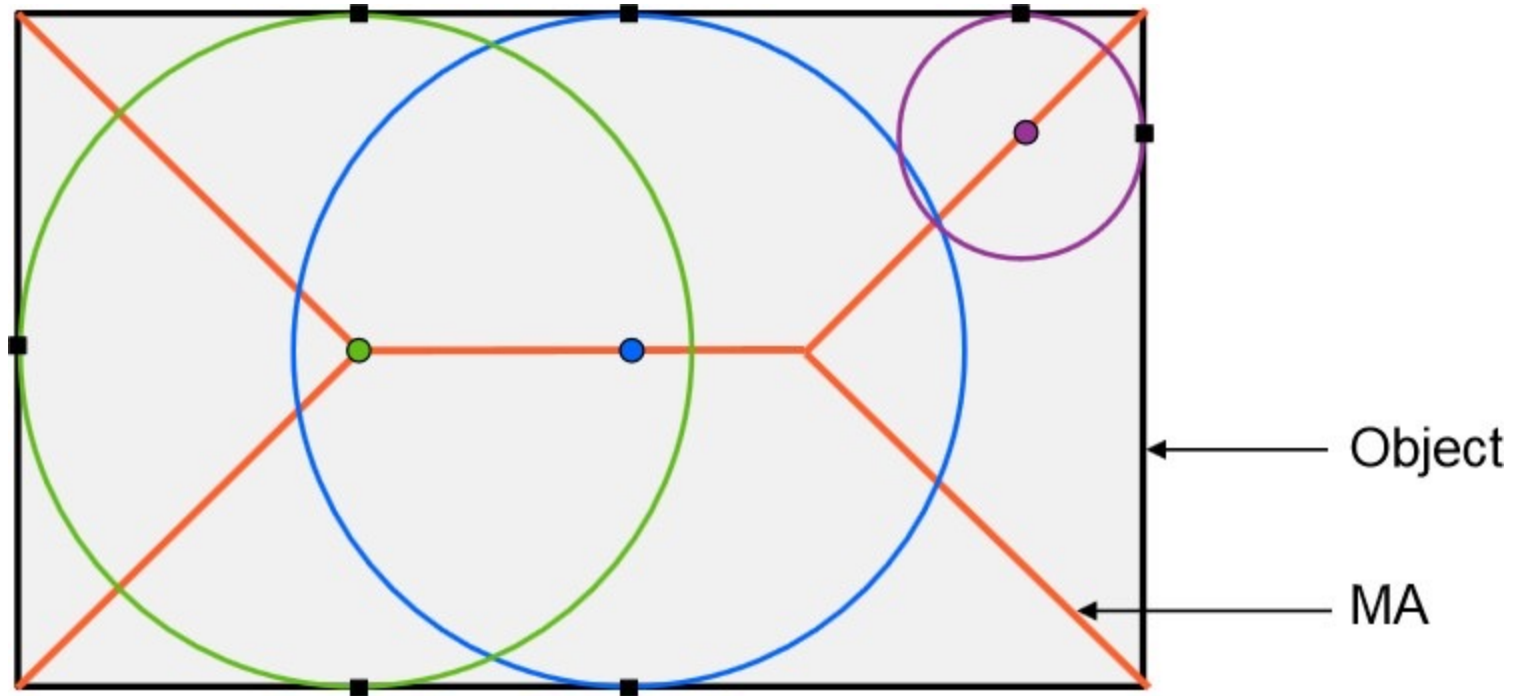


The Medial Axis

- **The Medial axis**: the locus of the centers of circles that are tangent to the polygon at two or more points.
- **locus**: a set of points whose location satisfies one or more specified conditions.

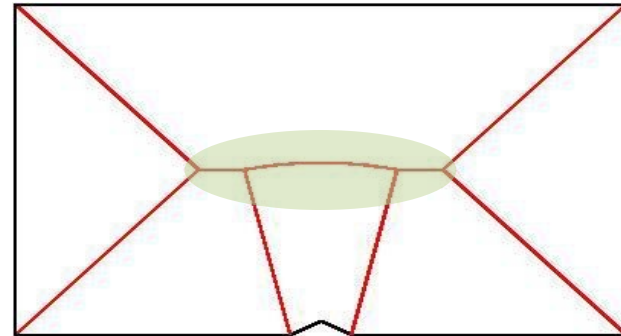
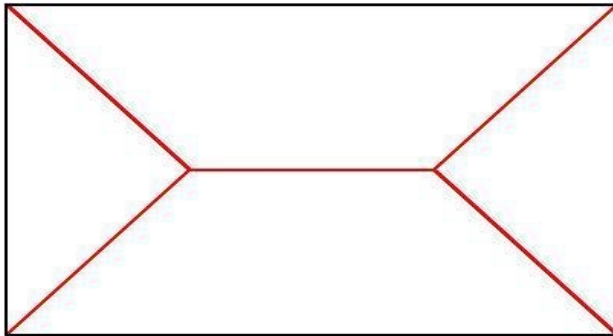


The Medial Axis: Example



The Medial Axis: Continued

- The Medial Axis comprises straight lines if the polygon is convex.
- If the object is concave, every reflex vertex induces a curved edge.

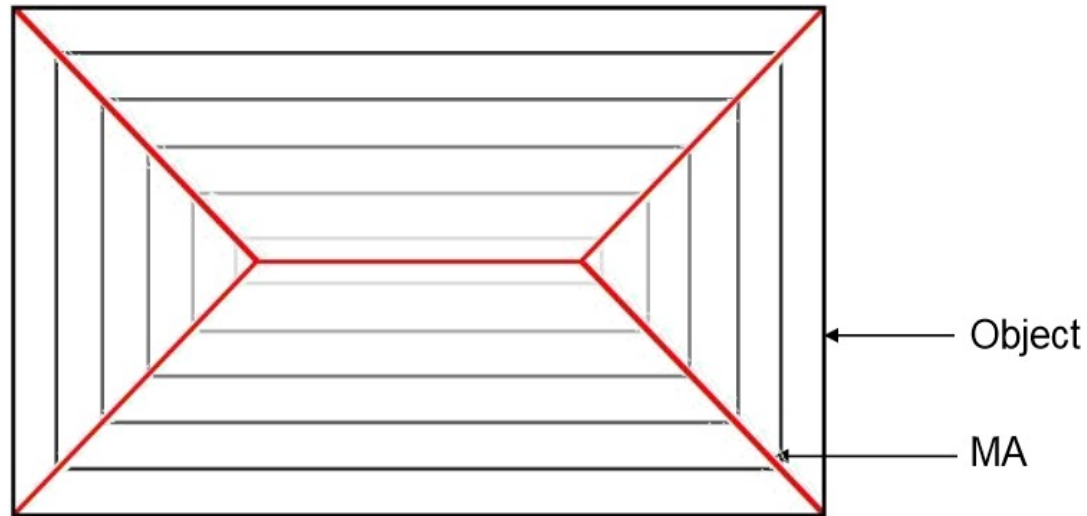


The Straight Skeleton

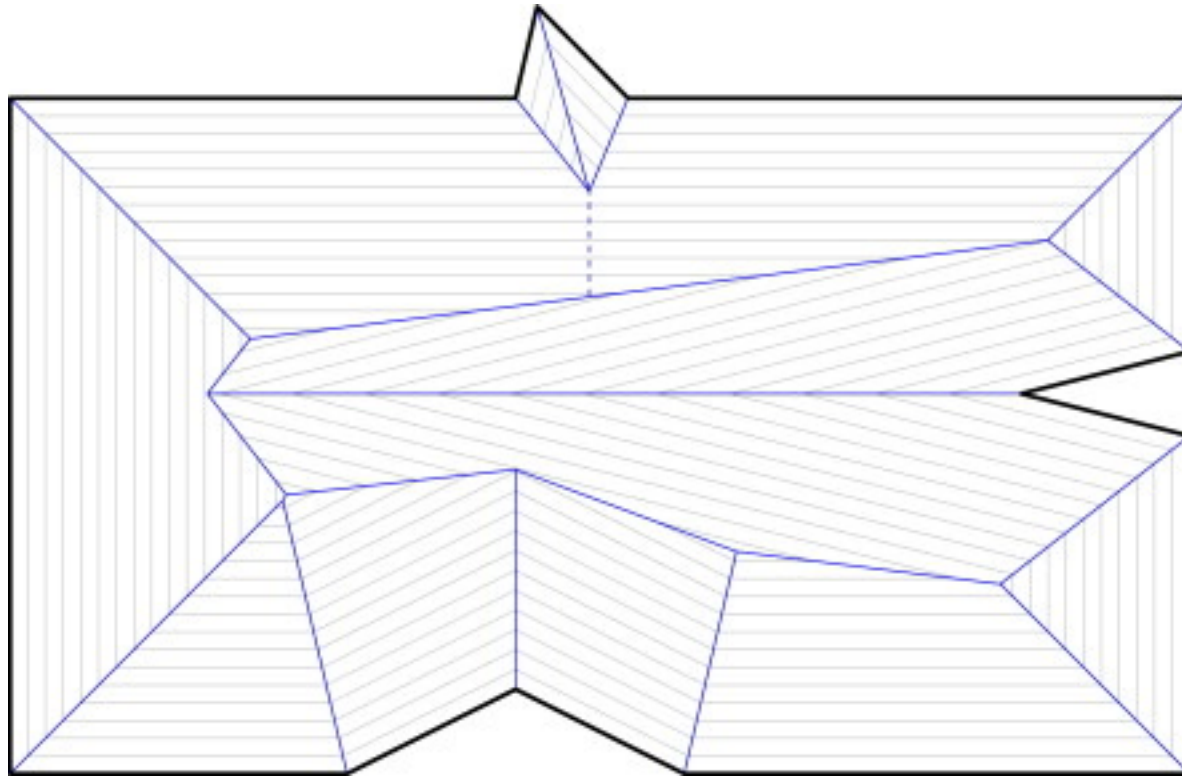


The Straight Skeleton

- The Straight Skeleton: the **trace** of the *angular bisectors* of the vertices, as the edges of the polygon are propagating at equal rate, until the polygon vanishes.
- It is a **linear approximation** of the Medial Axis.



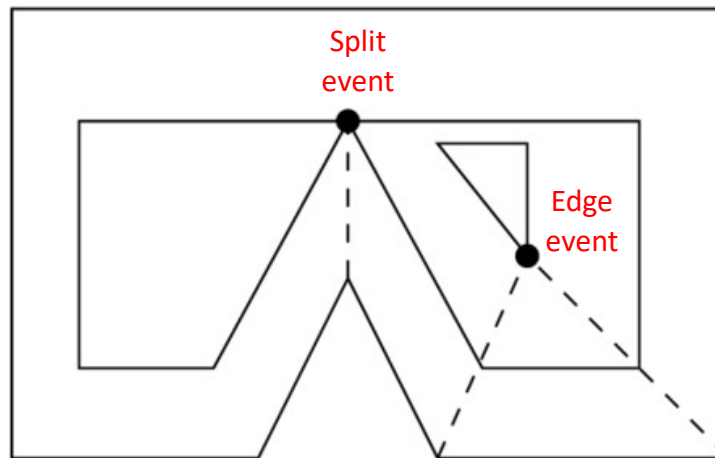
The Straight Skeleton



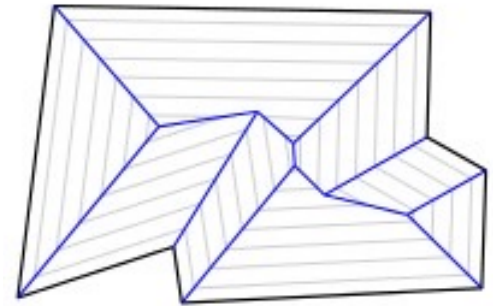
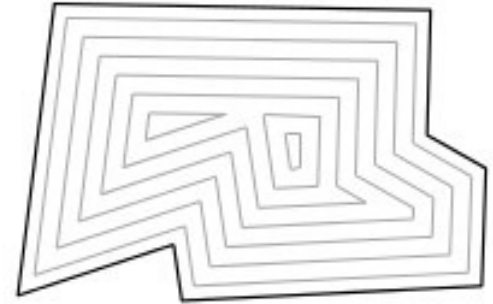
The Propagation of The Polygon

Two possible events (assuming g.p.) may occur during the propagation:

- **Edge Event**– A portion (or the whole) of an edge vanishes.
- **Split Event**– A reflex vertex hits an opposite edge, splitting the polygon into two disconnected parts.

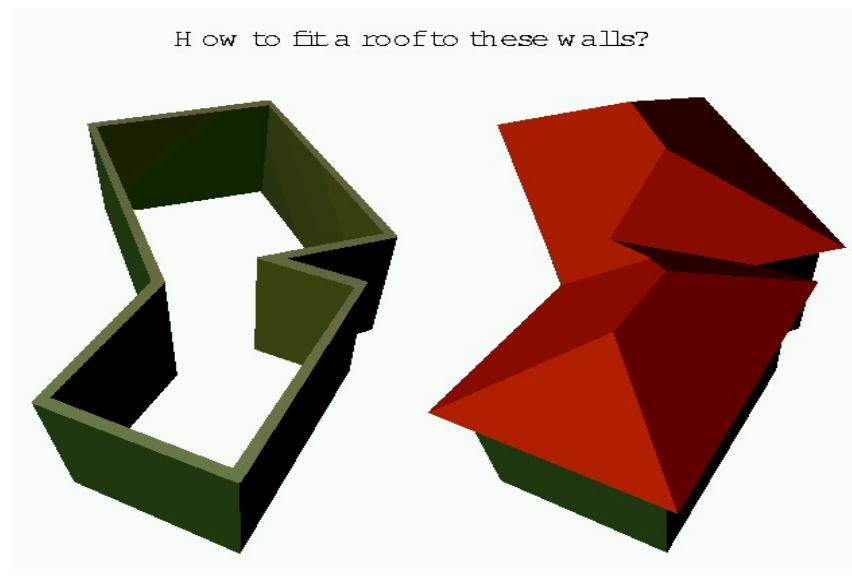


An Application of The Straight Skeleton



Designing Rooftops

- When assigning a height field to an inner node - its offset distance from the edge - the skeleton can be interpreted as the rooftop of a house which walls are the sides of the original polygon.



The Properties of The Straight Skeleton

- The faces of the straight skeleton are monotone (why?).

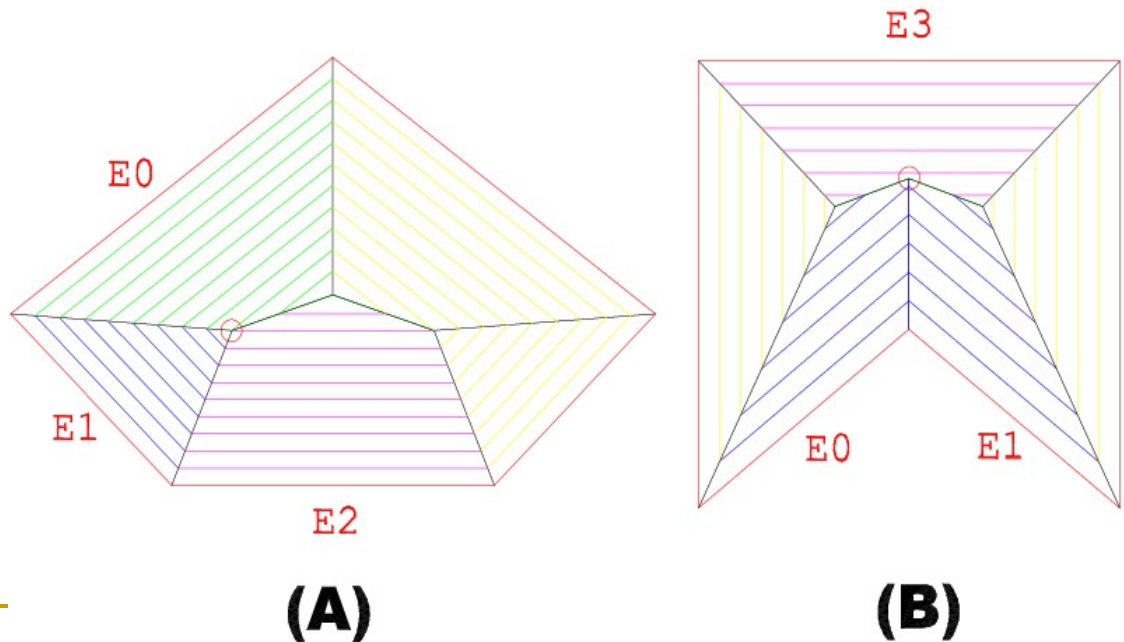


Figure 2

The Properties of The Straight Skeleton

- The faces of the straight skeleton are monotone (why?).
 - Every internal skeleton node has degree 3*
 - The Medial Axis and the straight skeleton of a convex polygon are identical.
 - There are $2n - 3$ edges, $n - 2$ inner vertices and n faces in a straight skeleton.
-

Straight-Skeleton Computation

- Most algorithms take a straight-forward approach of event-based simulation of the propagation.
- The most time-efficient algorithm known has time complexity $O(n^{1+\varepsilon} + n^{8/11+\varepsilon} r^{9/11+\varepsilon})$

r =# reflex vertices

n =# vertices

Felkel & Obdrzalek 98'

- Felkel & Obdrzálek offered a straightforward **event-based** algorithm.
 - The algorithm computes and simulates the events by maintaining a set of circular **Lists of Active Vertices** called **LAVs**.
 - The algorithm does not construct the intermediate offset polygons (although easily deduced), but only the skeleton itself.
-

The Algorithm for Convex Polygons



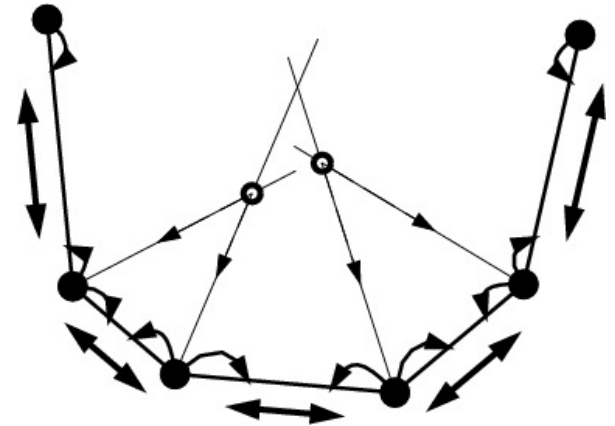
The Algorithm for Convex Polygons

■ Initialization

- Create a **LAV** for the polygon – a circular list of its vertices by order.
- Add pointers for the edges between vertices.
- Compute a bisector per vertex.
- All vertices are marked “*unused*”.

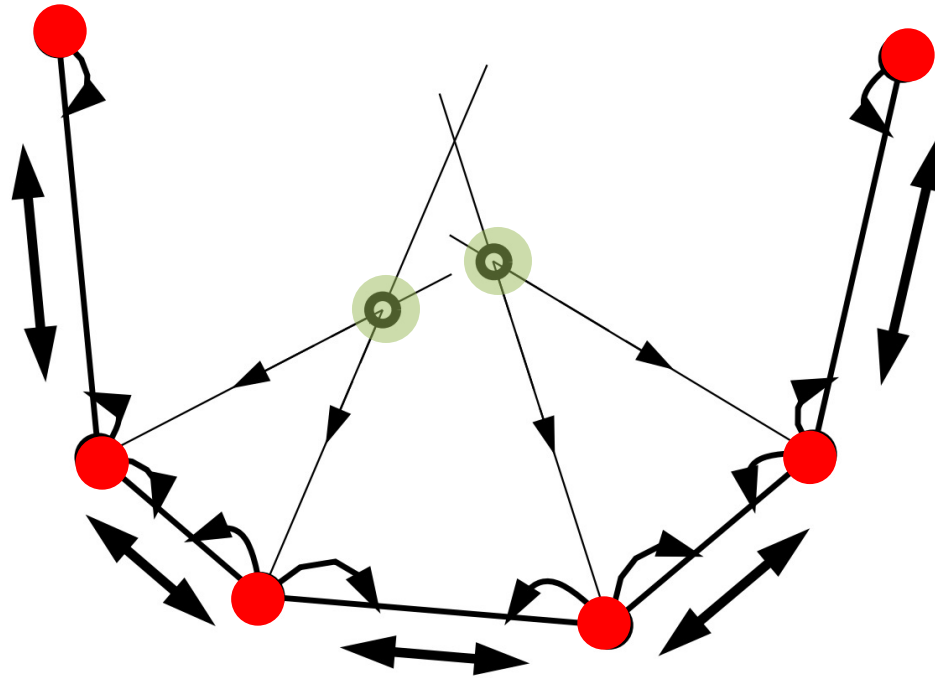
■ Calculation of initial edge events

- Compute the intersection point of every set of adjacent bisectors – this point is the location of the edge event between them.
- Queue the edge event (marked `EDGE_EVENT`) in a priority queue according to the distance of the intersection from the line supporting the edge.



Propagation

- = active vertices/nodes in the list (LAV)
- ⊗ = vertices/nodes marked as processed
- = intersection points in the priority queue
- = the current node V
- ↔ = pointers in the list of active vertices (LAV)
- ↷ = pointer to the appropriate edge for bisector computation



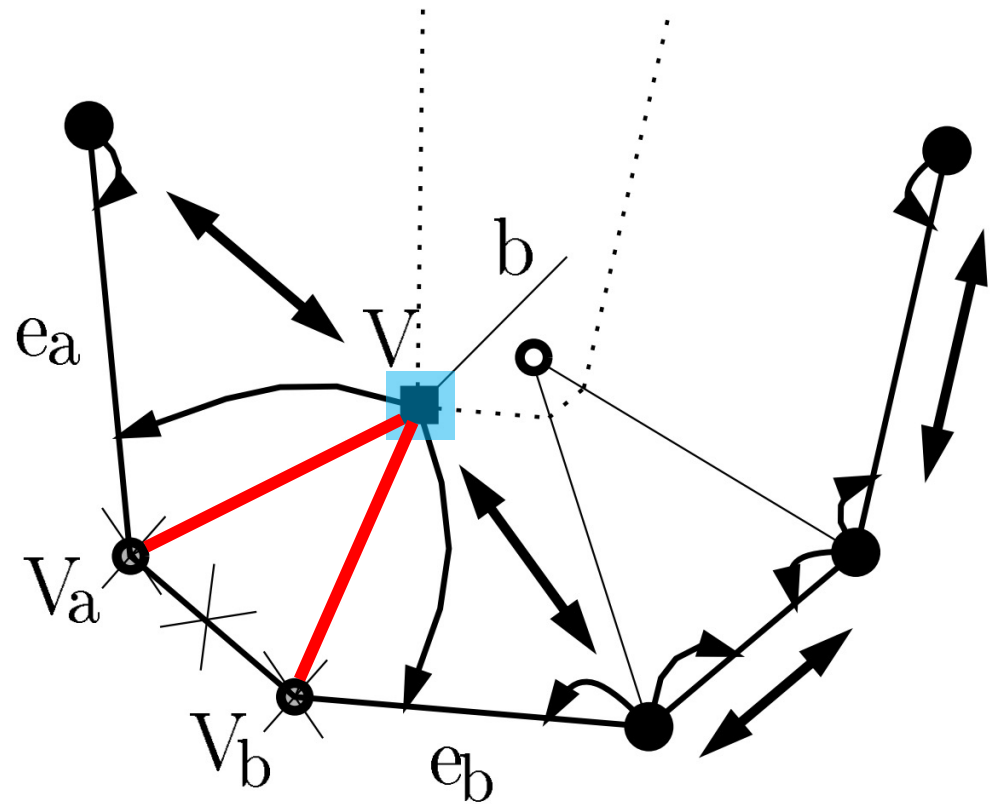
Propagation Step

- While the events queue \neq empty do
 - If next event uses used vertices, discard event.
 - Else, handle edge event
 - If LAV contains more than 3 edges
 - Create two edges of the skeleton, each from one of the event vertices to the location of the event (the intersection point).
 - Remove these two vertices from the LAV, and mark them as “used”.
 - Create a new vertex, located at the intersection point, and put it in its place in the LAV, pointing to its adjacent edges.
 - Compute new edge events for the vertices of these adjacent edges.
 - Else, create new vertex at the intersection, and skeletal edges from each of the 3 vertices.

Propagation

- = active vertices/nodes in the list (LAV)
- ⊗ = vertices/nodes marked as processed
- = intersection points in the priority queue
- = the current node V
- ↔ = pointers in the list of active vertices (LAV)
- ↷ = pointer to the appropriate edge for bisector computation

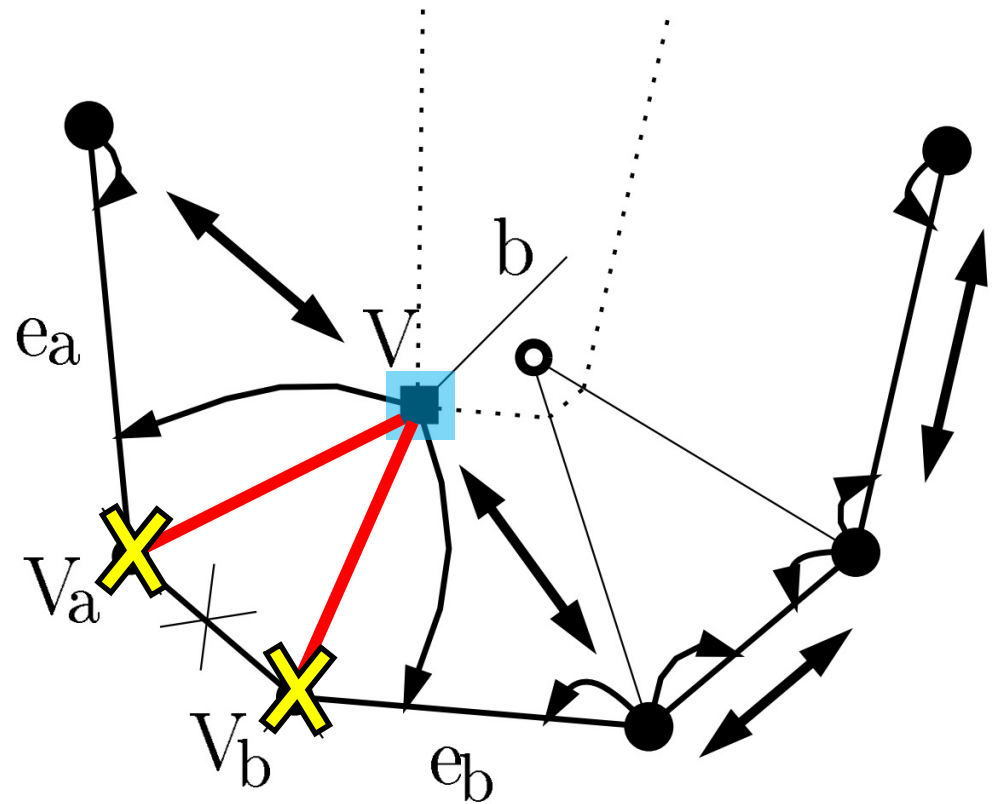
- Create two edges of the skeleton, each from one of the event vertices to the location of the event (the intersection point).



Propagation

- = active vertices/nodes in the list (LAV)
- ⊗ = vertices/nodes marked as processed
- = intersection points in the priority queue
- = the current node V
- ↔ = pointers in the list of active vertices (LAV)
- ↷ = pointer to the appropriate edge for bisector computation

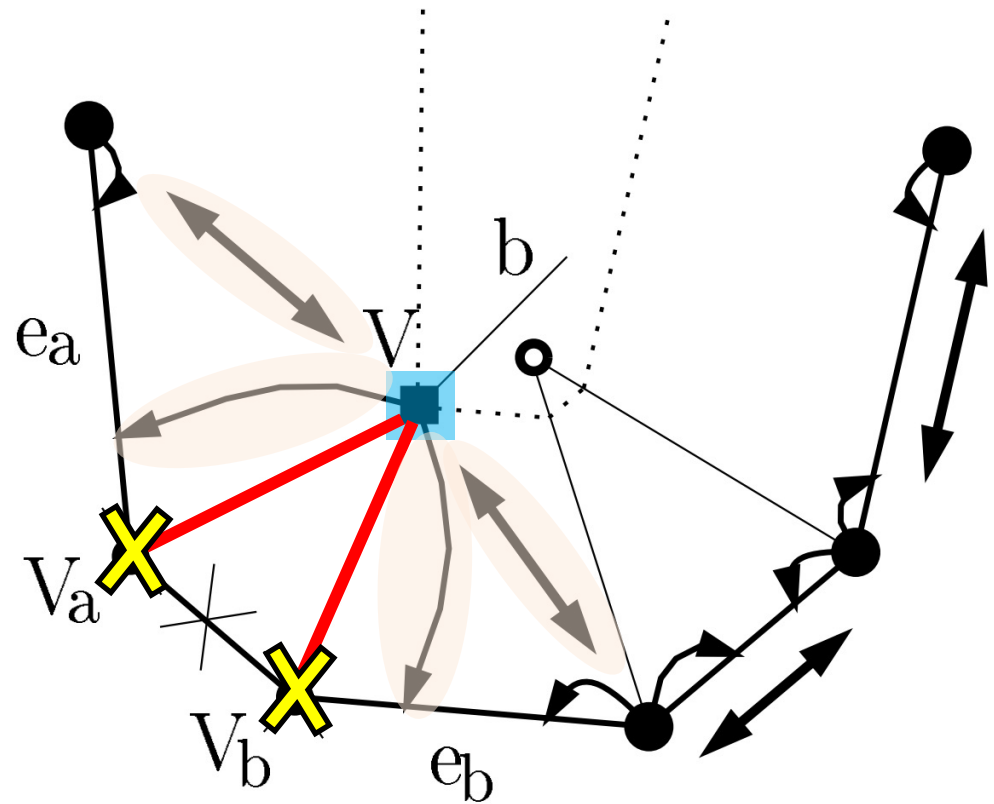
- Remove these two vertices from the LAV, and mark them as “used”.



Propagation

- = active vertices/nodes in the list (LAV)
- ⊗ = vertices/nodes marked as processed
- = intersection points in the priority queue
- = the current node V
- ↔ = pointers in the list of active vertices (LAV)
- ↷ = pointer to the appropriate edge for bisector computation

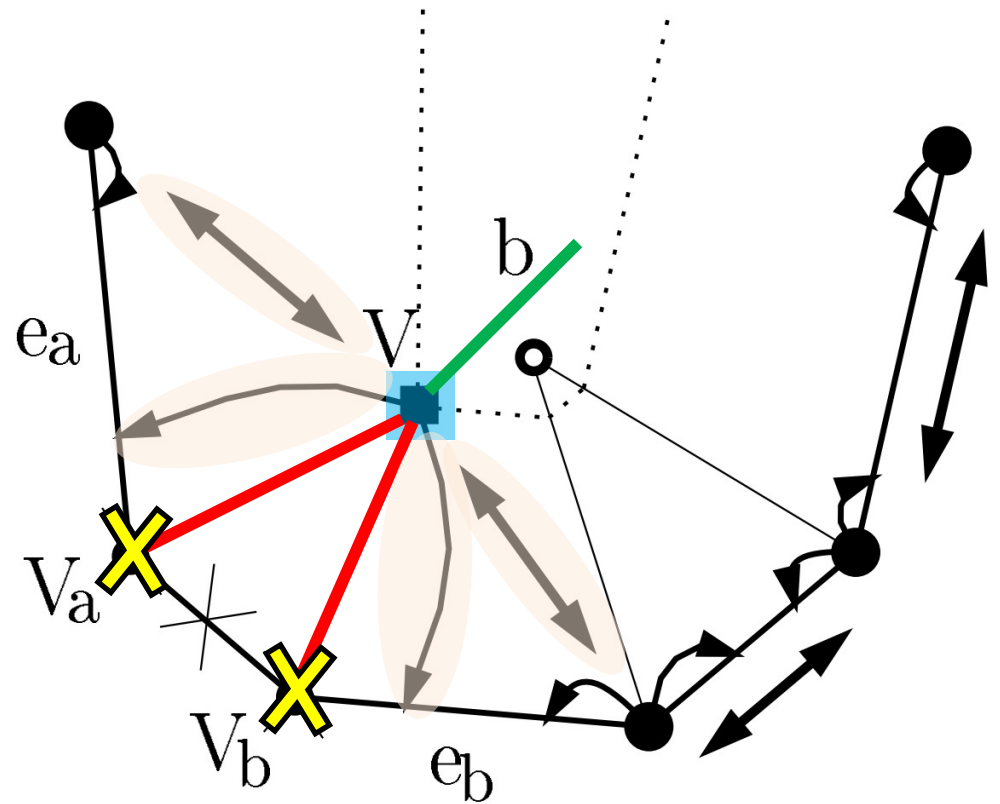
- Create a new vertex, located at the intersection point, and put it in its place in the LAV, pointing to its adjacent edges.



Propagation

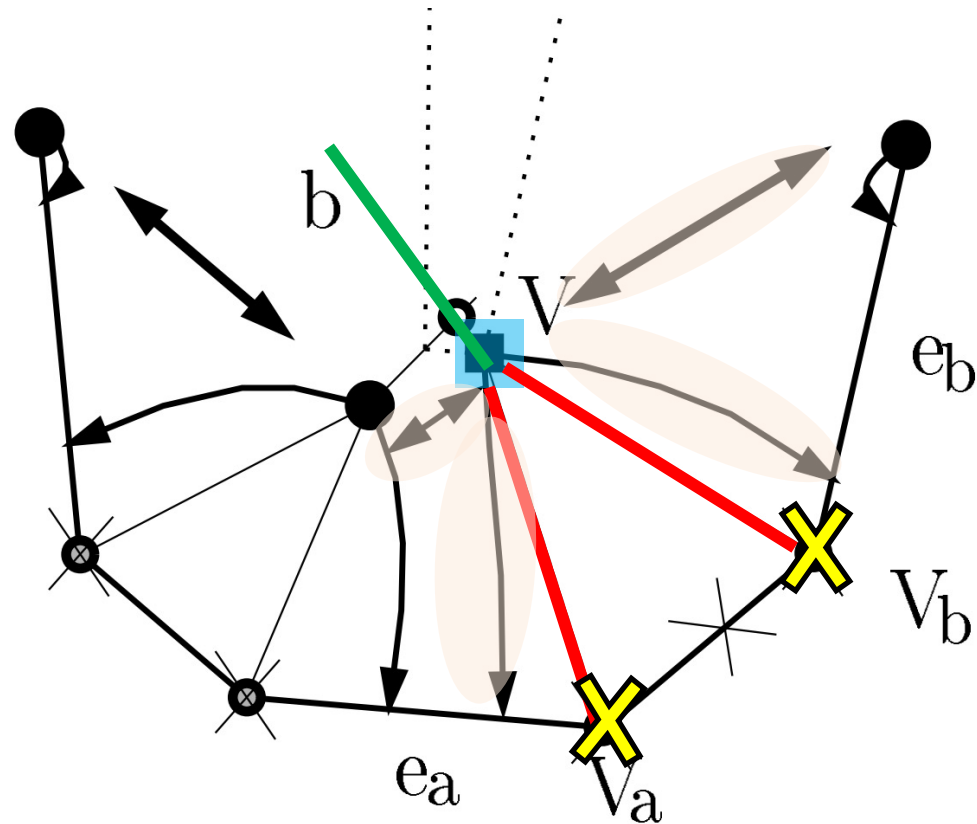
- = active vertices/nodes in the list (LAV)
- ⊗ = vertices/nodes marked as processed
- = intersection points in the priority queue
- = the current node V
- ↔ = pointers in the list of active vertices (LAV)
- ↷ = pointer to the appropriate edge for bisector computation

- Compute new edge events for the vertices of these adjacent edges.



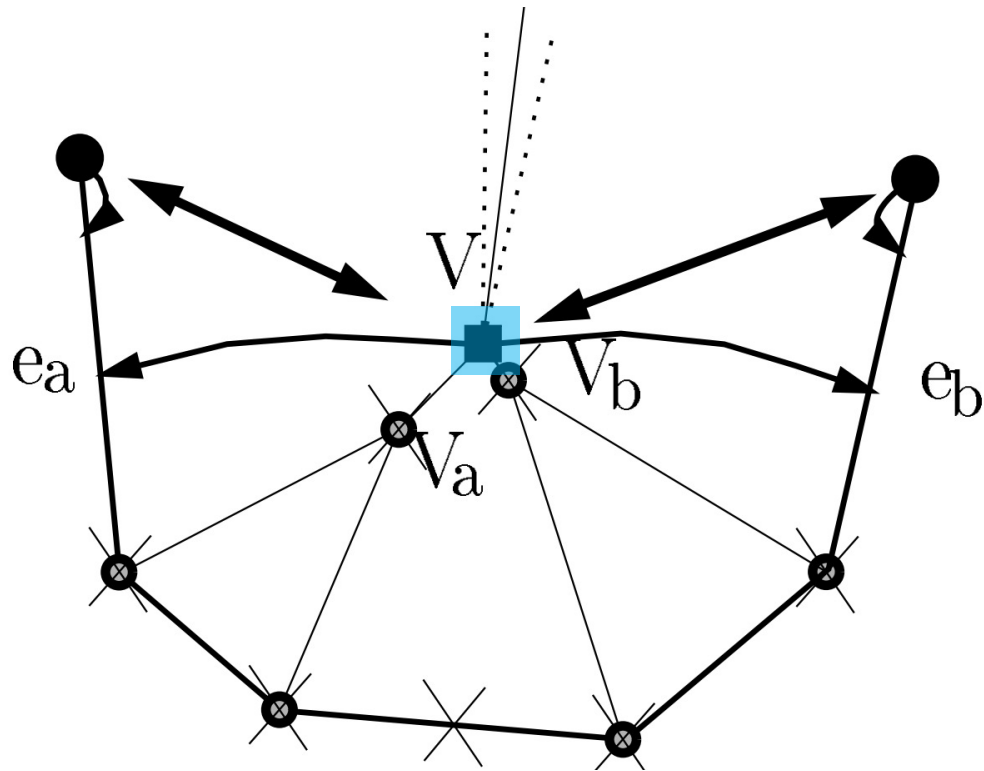
Propagation

- = active vertices/nodes in the list (LAV)
- ⊗ = vertices/nodes marked as processed
- = intersection points in the priority queue
- = the current node V
- ↔ = pointers in the list of active vertices (LAV)
- ↷ = pointer to the appropriate edge for bisector computation



Propagation

- = active vertices/nodes in the list (LAV)
- ⊗ = vertices/nodes marked as processed
- = intersection points in the priority queue
- = the current node V
- ↔ = pointers in the list of active vertices (LAV)
- ↷ = pointer to the appropriate edge for bisector computation



Complexity

- The number of vertices reduces to zero, and the algorithm always stops.
 - Complexity: $O(n \log n)$, for maintaining the events queue. Every event handling is $O(1)$.
-

The Algorithm for Nonconvex Polygons

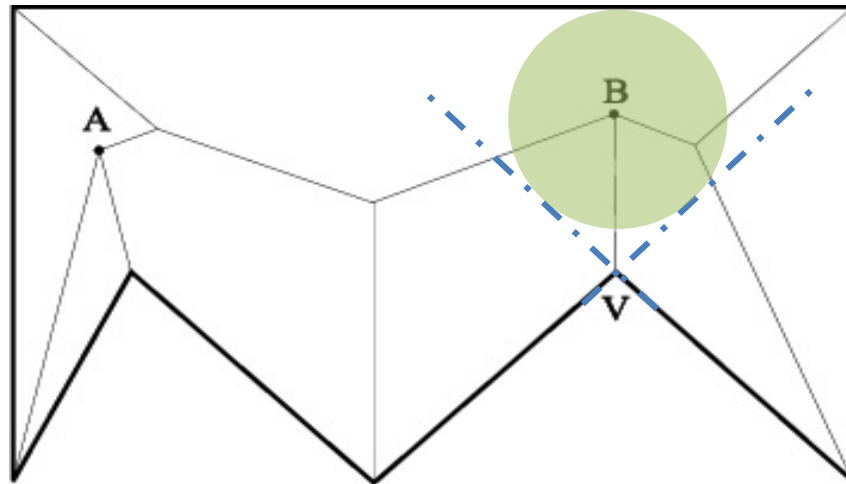


The Algorithm for Nonconvex Polygons

- An extension of the convex algorithm.
 - We have to find out when split events occur.
 - Another step in initialization:
 - Determine all possibilities of a reflex vertex hitting an opposite edge.
 - Queue these events as **SPLIT_EVENT**
-

Obtaining Split Events

- A splitting location B is equidistant from:
 - the lines supporting the edges adjacent to the reflex vertex, and;
 - the line supporting the opposite edge.

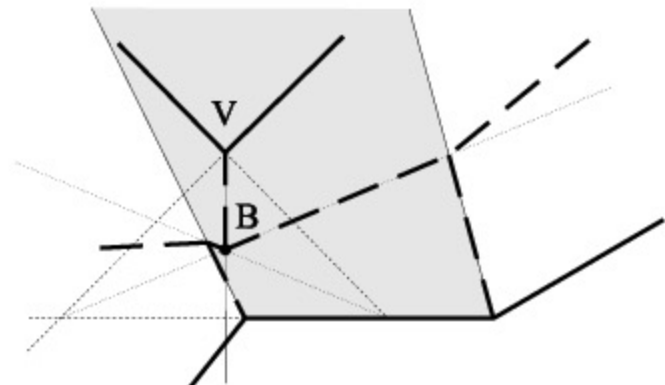
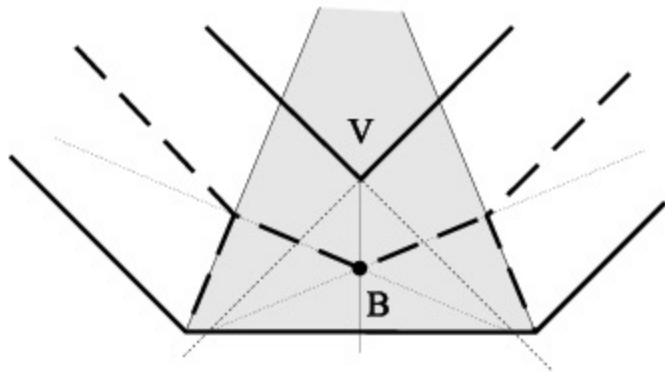


Obtaining Split Events

- A splitting location B is equidistant from:
 - the lines supporting the edges adjacent to the reflex vertex, and;
 - the line supporting the opposite edge.
 - For every reflex vertex, we traverse all of the edges in the polygon and test for intersection.
 - A simple intersection test between the bisector of the reflex vertex and the opposite edge isn't enough (why?).
-

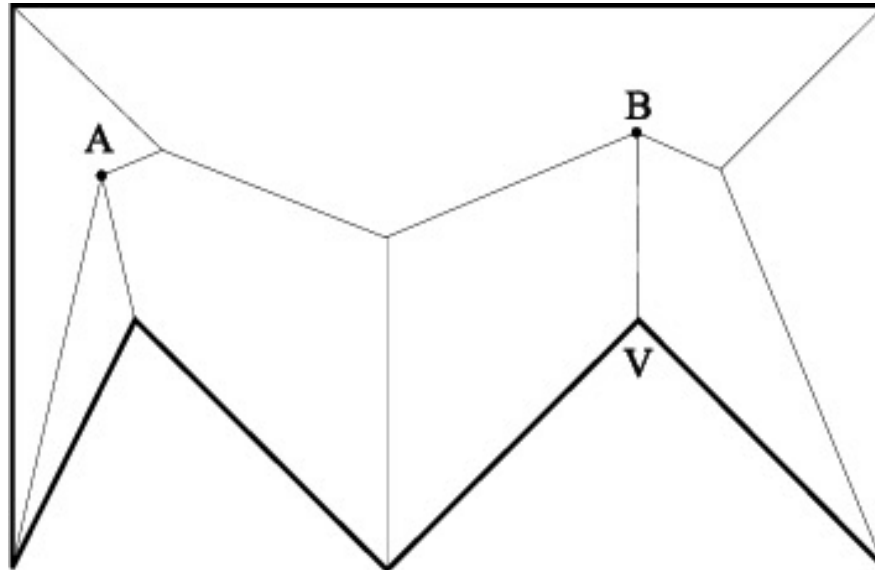
Obtaining Split Events - Continued

- The intersection point between the reflex vertex and the line supporting the opposite edges must be in the area defined between the edge and the bisectors of its two vertices.
- The intersection point is the meeting point of the three bisectors between all three participating edges (the two defining the reflex vertex and the split edge).



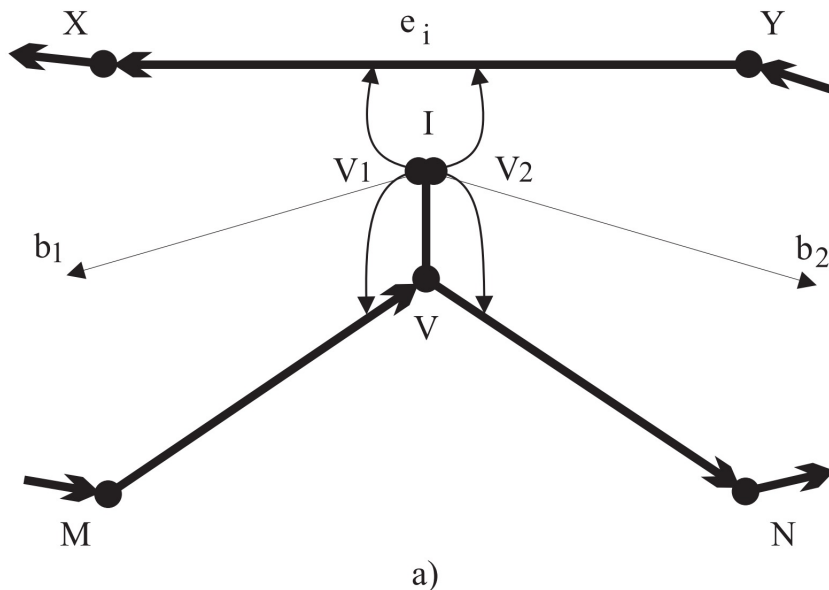
Obtaining Split Events

- Not all reflex vertices eventually cause split events. (A is an edge event, and B is a split event).



Handling Split Events

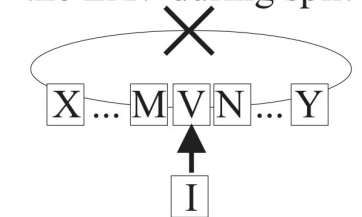
- When a split event occurs, the polygon splits into two parts.
- The LAV in context is split into two LAVs as well.



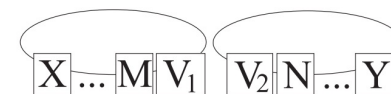
the LAV before split



the LAV during split



the LAVs after split



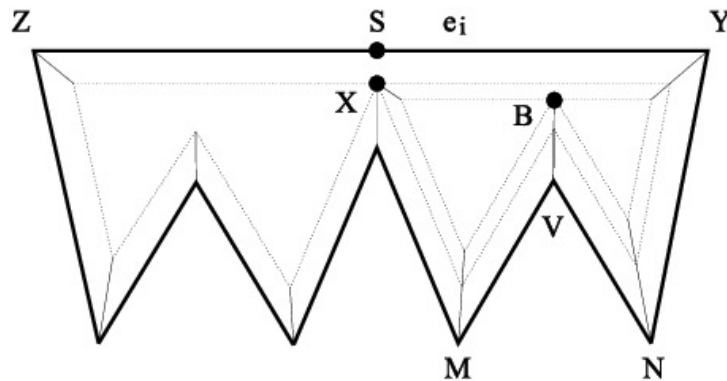
b)

Handling Split Events – Cont'd

- The splitting vertex is replaced with two new vertices, each in the appropriate place in a different LAV.
 - New bisectors and edge events are calculated for each of these vertices (why only edge events?)
 - The propagation continues...
-

Handling Multiple Splitting

- An edge can be split several times.
- Any split event handling must realize what part of the edge it is splitting (i.e. what are the proper endpoints).
- It is done by traversing the LAV in context at each handling of a split event.



Summary of the General Algorithm

■ **Initialization**

- Create one LAV
 - Compute bisectors
 - Compute split and edge events
 - Queue all events according to time (distance)
-

Summary – Continued

■ Propagation

- While event queue has events
 - If new event contains used vertices, discard event.
 - If event is edge event, handle as in the convex case. Mark vertices as “used”. If the LAV in context contains 3 vertices, close up the skeleton.
 - If event is split event, split the LAV into two, and maintain pointers accordingly. Mark the splitting vertex as “used”.

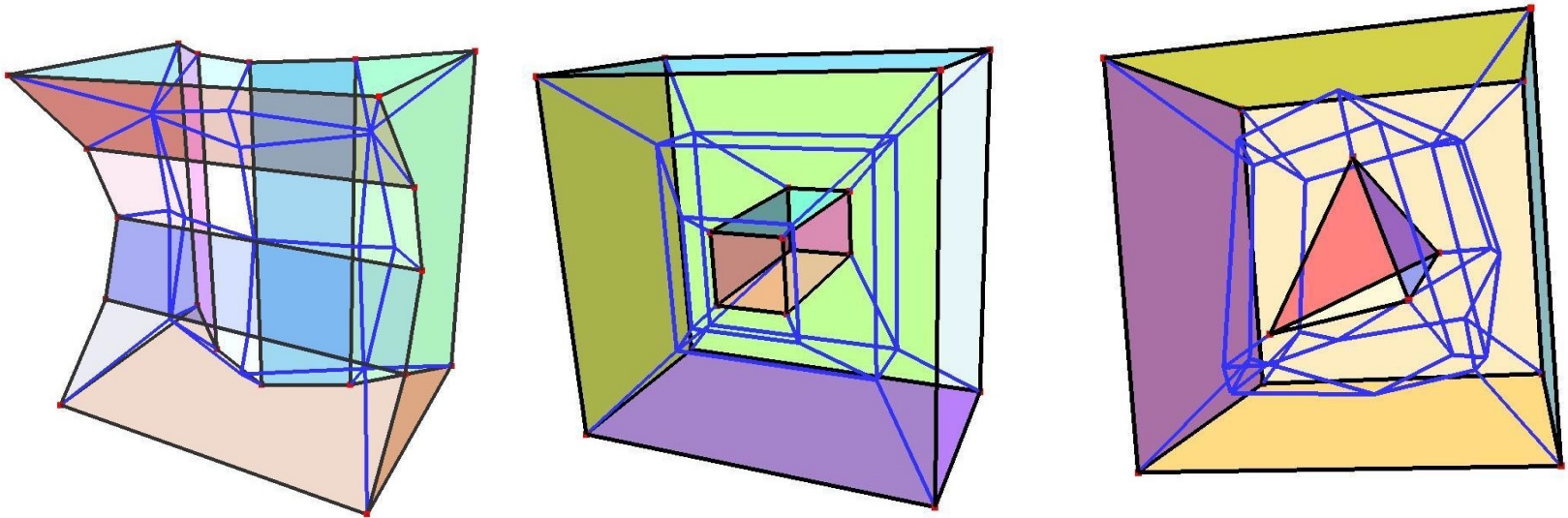
- In the end, there are no LAVs left!
-

The Complexity of The Algorithm

- Initializing and handling each split event require traversing all of the edges per each reflex vertex.
 - So, the total complexity is $O(rn + n \log n)$.
 - $r = \#$ reflex vertices
 - If $r = O(n)$ then the algorithm is quadratic – a reachable upper bound.
 - Most practical cases behave better.
 - Space complexity is $O(n)$.
-

3D Straight Skeleton – A (New) View

- The faces of a polyhedron propagate at equal rate.
- Skeleton is the trace of faces, edges and vertices.



Bibliography

■ Source of images (and recommended reading):

- “Medial Axis presentation” -
<http://groups.csail.mit.edu/graphics/classes/6.838/F01/lectures/MedialAxisEtc/presentation/>
- “Single-Fold Disk Hiding” -
<http://jeff.cs.mcgill.ca/~mcleish/507/single.html>
- “Straight skeleton of a simple polygon” -
<http://compgeom.cs.uiuc.edu/~jeffe/open/skeleton.html>
- “Raising roofs, crashing cycles, and playing pool” -
<http://compgeom.cs.uiuc.edu/~jeffe/pubs/cycles.html>
- “Designing Roofs of Buildings “ -
<http://www.sable.mcgill.ca/~dbelan2/roofs/roofs.html>

■ Straight Skeleton Computation

- P. Felkel and S. Obdrzalek, Straight skeleton computation, Spring Conf. on Computer Graphics, Budmerice, Slovakia, 210--218, 1998.
-