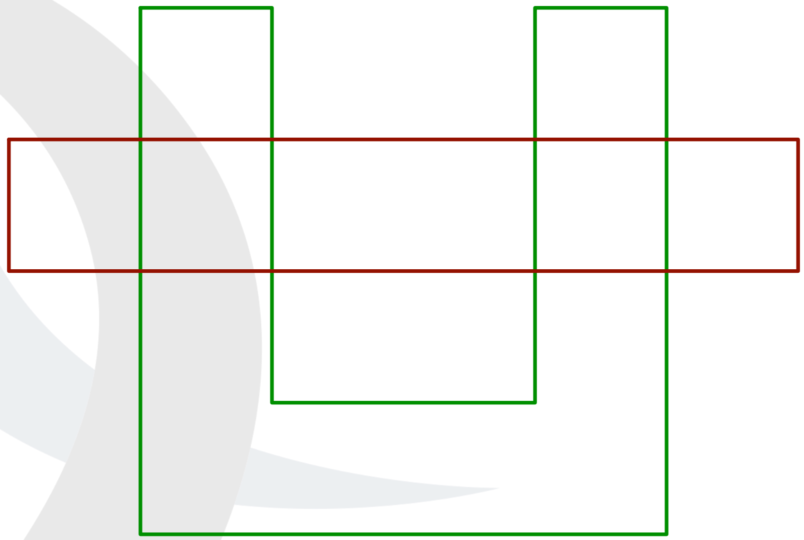
Polygon Merge
Presentation Excerpts from NFM21

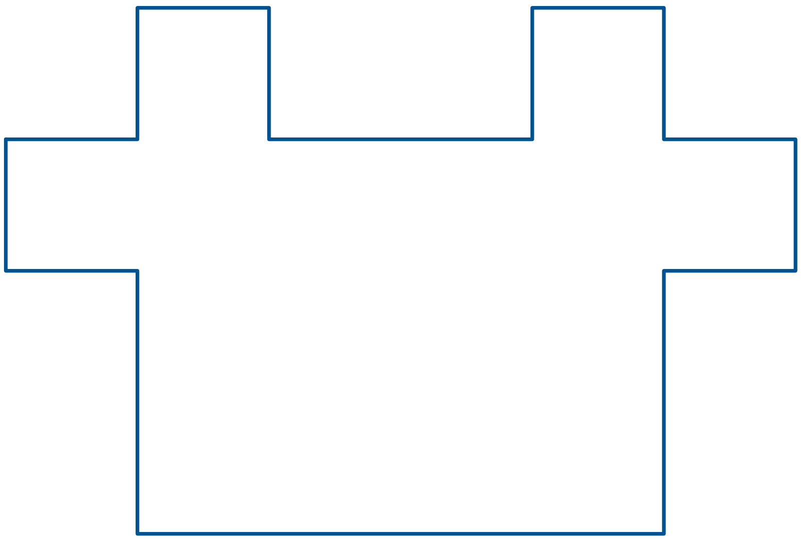
- We have developed an algorithm to compute a “merged” polygon given two overlapping input polygons A and B
- Anticipated application is computing keep-in and keep-out zones for autonomous vehicles
- Work is funded by Air Force Research Laboratory
- We have formalized and verified the algorithm using PVS
- Required a surprisingly large effort resulting in many intermediate definitions and lemmas

Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

Motivating Example



A merge B



C = A merge B

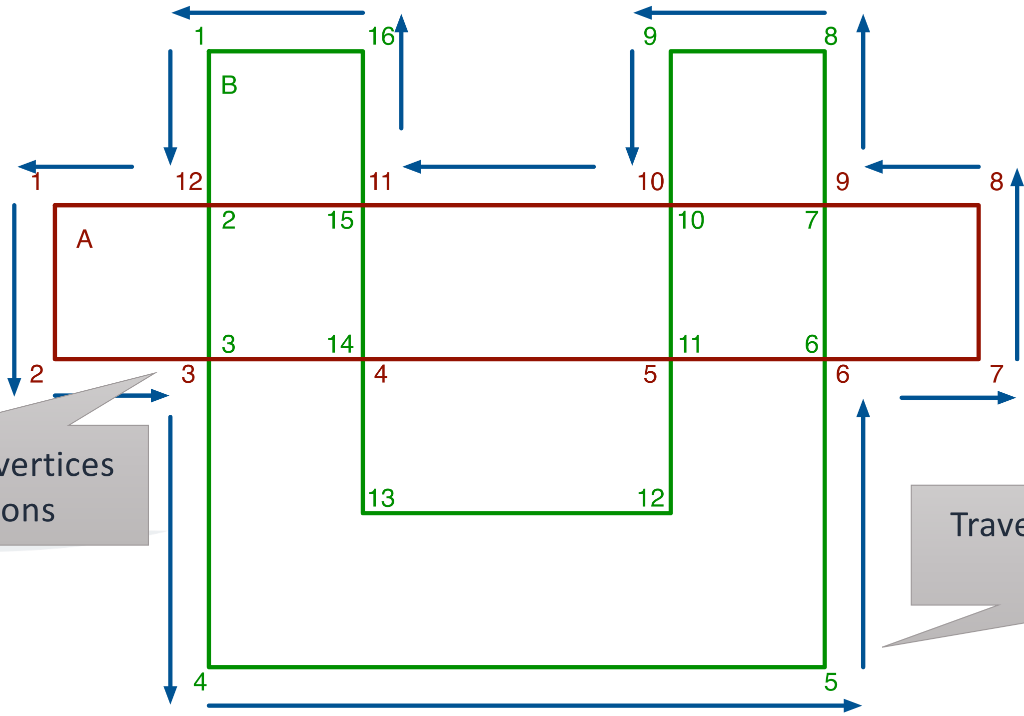
Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

Polygon Merge Algorithm

- Algorithm starts by creating derived polygons Am and Bm by "injecting" intersection vertices into polygons A and B
- Next, a traversal of the vertices is used to identify the outermost edges of Am and Bm
- Traversal starts at the topmost of the leftmost vertices of Am and Bm
- Terminates when traversal returns to starting vertex
- We assume simple polygons having counterclockwise vertex order

Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

Algorithm: Inject Vertices and Traverse Outside Edges



Inject intersecting vertices into both polygons

Traverse outermost edges to produce the merge

Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

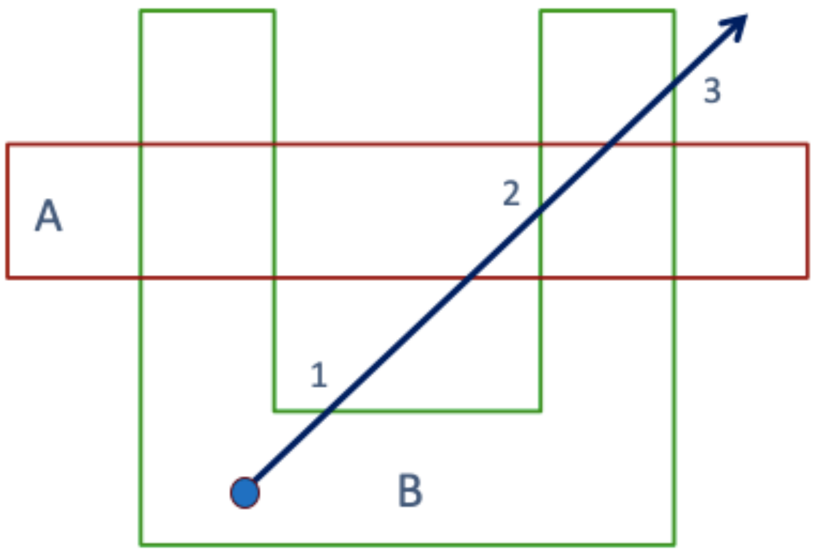
Polygon Merge Requirements

- Merge algorithm creates a polygon C that “contains” both polygons A and B
- Show that algorithm implies *point-set membership* properties
- Need to establish that all points in A and B are also in C
- Also need to show the converse, that the containment is “tight”
- A point in C must appear in A or B or one of the “holes”
- Zero or more holes exist for given A and B (regions completely surrounded by edges of A and B)
- Two main theorems have been proved to show these results hold

Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

Point Membership Criterion

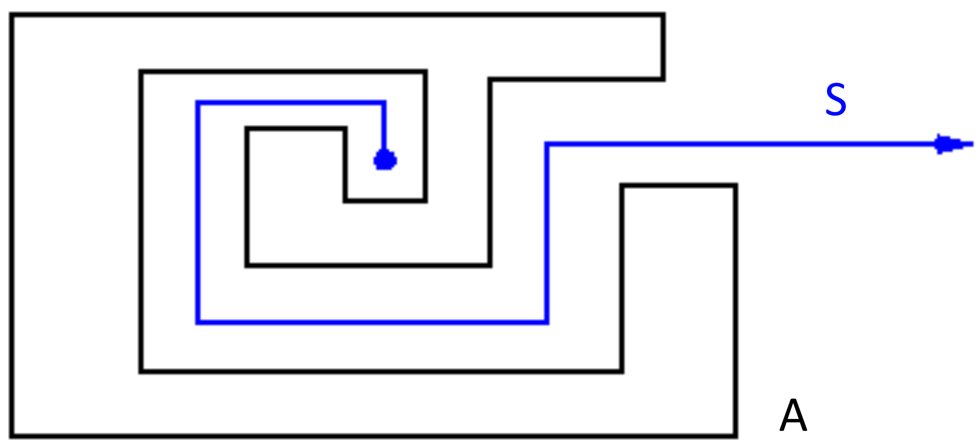
- Determining whether a point lies within a polygon
- Cast a ray in any direction and count the number of crossings
- Odd number means point is inside; even means outside
- Point is outside of A and inside of B (2 vs 3 crossings)
- Also, horizontal ray version



Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

Additional Membership Criterion

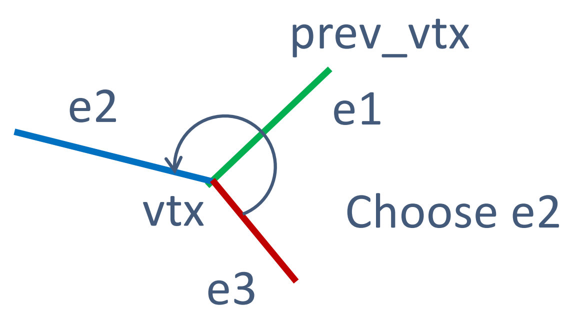
- Added a non-computable way to check if point is outside
- Introduced concept of a *serpentine ray*
- A hybrid geometric object
- Head: point, body: connected line segments, tail: ray
- Point outside: there exists an S such that no point on S intersects any edge of A



Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

Algorithm Detail
Finding Rightmost Edge

- Merge algorithm selects rightmost edge when it has a choice of two
  - One from Am, one from Bm
- CCW traversal ensures rightmost will be outermost
- Relies on predicates to test if a point lies between two edges (in angular sense)
- E.g., prev_vtx between e3 & e2




```
between_rays?(s, e: segment_2d)
                   (p: point_2d): bool =
  LET u = s`p1, v = e`p1, w = e`p2,
      uv = u - v, pv = p - v, wv = w - v IN
   IF cross(uv, wv) < 0
     THEN  cross(uv, pv) > 0 OR cross(wv, pv) < 0
      ELSE  cross(uv, pv) > 0 AND cross(wv, pv) < 0
  ENDIF
```

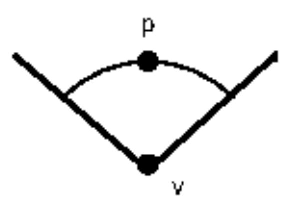

Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

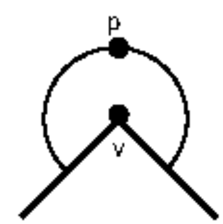
Vertex Wedge Regions

- We bound the wedge region in front of a vertex using a circular arc at a fixed distance
- Added a predicate to say when a point is within wedge
- Concept used in proofs to show no other edges pass through wedges

```
vertex_wedge_radius(G: simple_polygon_2d): posreal =
   min_edge_sep(G) / 2

point_in_vertex_wedge?(outward: bool,
                                        G: (ccw_vertex_order?),
                                         i: below(G`num_vertices))
                                       (p: point_2d): bool =
  point_between_edges?(outward, G, i)(p) AND
  norm(p - G`vertices(i)) <= vertex_wedge_radius(G)
```





Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

Forward Point Containment

- First reduce point containment to edge containment
- Edges contained: any point on A’s perimeter must be:
  - Inside of C, or
  - On C’s perimeter
- Edge containment implies point containment

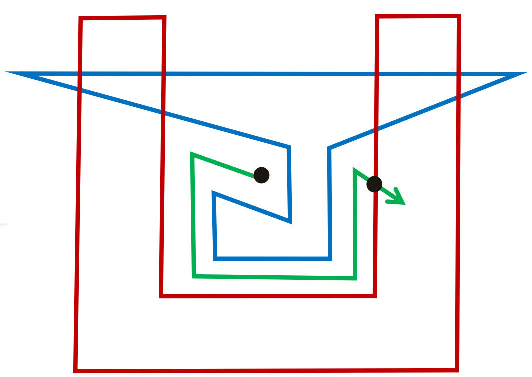
```
polygon_contained?(A, C: simple_polygon_2d): bool =
  FORALL (p: point_2d):
    point_on_polygon_perimeter?(A)(p)
      IMPLIES point_in_polygon?(p, C) OR
               point_on_polygon_perimeter?(C)(p)

contained_membership: THEOREM
  FORALL (p: point_2d, A, C: simple_polygon_2d):
    polygon_contained?(A, C) AND point_in_polygon?(p, A)
      IMPLIES point_in_polygon?(p, C)
```


Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

Reverse Point Containment

- Points in C must lie in A, B or a hole
- Definition for point in hole relies on serpentine ray concept
- Hole points have no escape path
- Every s-ray must cross an edge of A or B



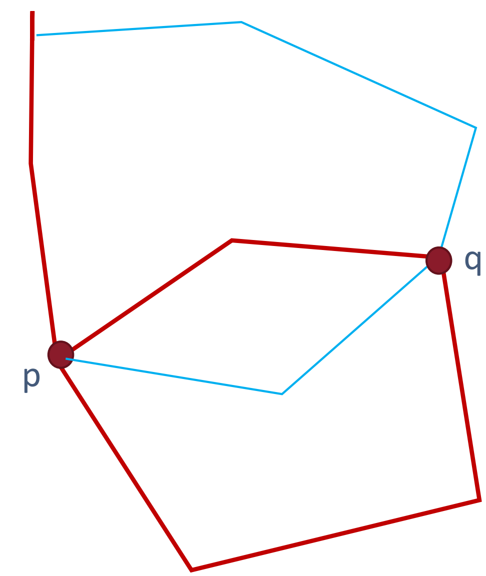
```
point_in_polygon_merge_hole?
        (A, B: simple_polygon_2d)
        (p: point_2d): bool =
  NOT point_in_polygon_inclusive?(p, A) AND
  NOT point_in_polygon_inclusive?(p, B) AND
  FORALL (S: serpentine_ray):
   p = S`body`seq(0) IMPLIES
     EXISTS (q: point_2d):
       point_on_serp_ray?(q, S) AND
       (point_on_polygon_perimeter?(A)(q) OR
        point_on_polygon_perimeter?(B)(q))
```


Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

Proving Uniqueness of Merge Vertices

- To be a simple polygon, merged polygon must have unique vertices
- Assumes vertex uniqueness for Am, Bm
- Need to rule out the "figure six" problem (prematurely looping back)

```
merge_seq_has_uniq_vertex_list: LEMMA
  FORALL (A: (ccw_vertex_order?),
             B: (ccw_merge_pre_condition(A))):
    LET M = merge_seq(A, B) IN
       uniq_vertex_list?(M`length)(M`seq)
```



Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

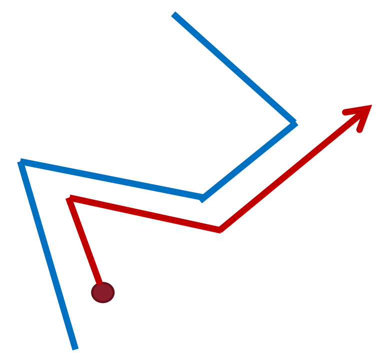
Proving Uniqueness of Merge Vertices

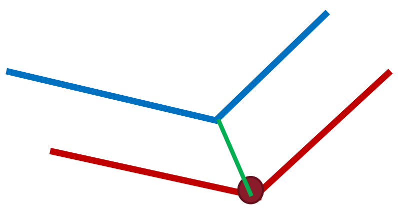
- Proving the uniqueness result has been one of the most difficult parts of the verification effort
- Difficulty stems from the need to work with a (partial) sequence of vertices rather than a complete polygon
- Many previously proved lemmas about polygons cannot be applied here
- Instead, we need to reason about a vertex sequence and a hypothetical sub-polygon created by it when the sequence folds back on itself

Funded by USAF AFRL/RQQA contract FA8650-17-F-2220.
Approved for Public Release (88ABW-2020-3765).

Point Membership
Edge-Skimming Paths

- We sometimes need to construct paths that "hug" edges without touching them
- Applies to sequences of line segments, not necessarily full polygons
- *Miter points* based on min values of:
  - Edge separation distances
  - Angles (sines) between edges
  - Edge lengths
- Lemmas added to support path reasoning





Miter point