This algorithm proceeds by shooting rays from each vertex to all the visible reflex vertices. If the resulting ray lands inside an open segment of δP, the intersection point is added to a vertex set v_new, which is then inserted into δP when all the ray shooting operations have completed.

Each reflex vertex induces a combinatorial change in the visibility polygon for points on either side of the resulting new vertex. This is because the originating vertex, from the original polygon, is visible on one side of the ray but not the other.

In pseudocode, the algorithm proceeds as

Algorithm 1 Partition the boundary of a polygon into visibility equivalence classes.

```
procedure PartitionPoly(poly)
    reflex_verts ← GetReflexVerts(poly)
    for v_r in reflex_verts do
        for v_vis in VisibleVerts(poly, v_r) do        ▷ vertices visible from reflex induce transitions
            v_new ← ShootRay(v_vis, v_r)
        end for
    end for
    P_ins ← InsertVerts(v_new, poly)
    return P_ins
end procedure
```

The resulting polygon, with v_new inserted, has the following property:

> The visibility polygons of any two points in the same open segment have the same combinatorial structure with respect to the original polygon P. The edge-edge visibility graph is the same for both visibility polygons, and there is a continuous transformation from one visibility polygon to another, without adding or deleting edges.

One notable condition of this property is that it holds only with respect to the original polygon P, not the new polygon P_{ins}.

But since we are using the edge-edge visibility properties of P_{ins} to do navigation, it might be nice to have this visibility-invariance property to hold for visible edges in P_{ins}.

Perhaps we can just iterate the original algorithm until convergence, at which point all segments will be guaranteed to have this property with respect to P_{ins}.

This appears to be possible for some polygons:

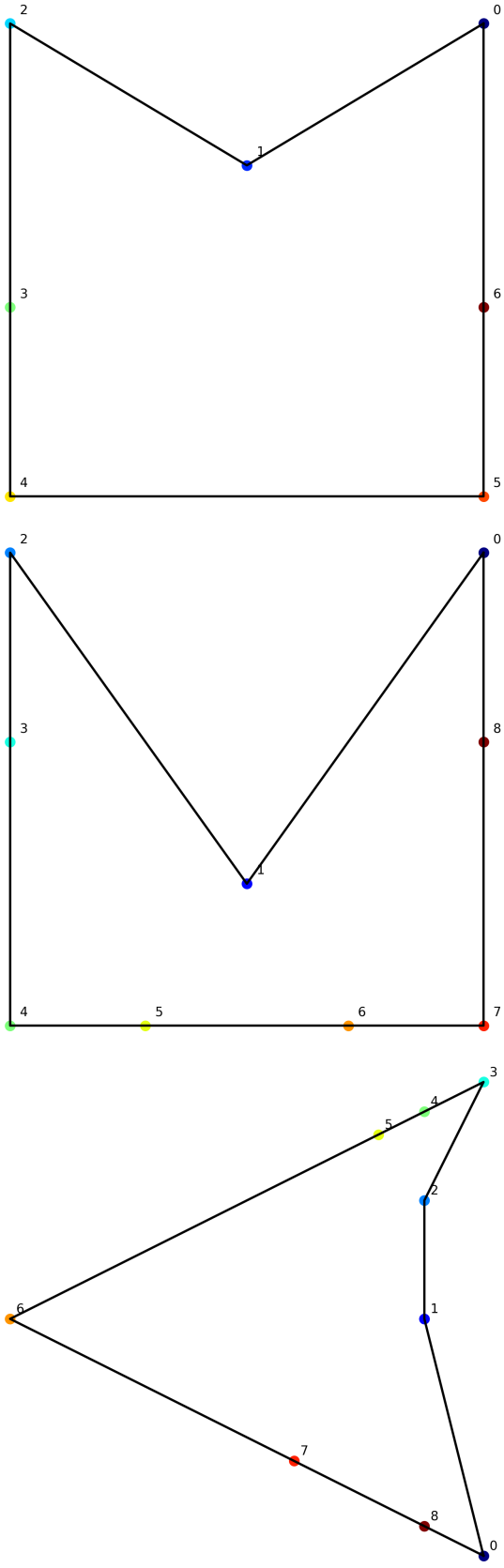

Figure 1: Polygons where the decomposition algorithm terminates in one iteration.

But other polygons do not seem to converge. We can keep iterating this example, and each iteration adds two more vertices (one for each reflex). At some point it hits the limits of machine precision and appears to converge. Interestingly, it appears to converge toward the line tangent to both reflex vertices (the bitangent).

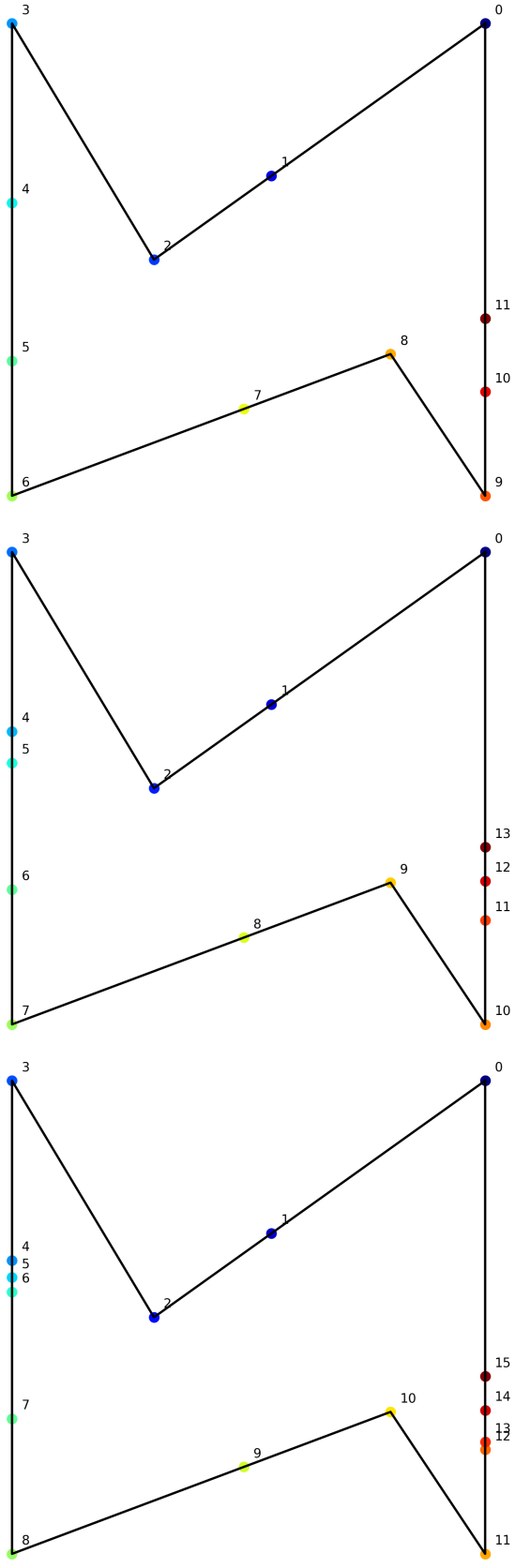
3
0
1
4
2
11
5
8
10
7
6
9
3
0
1
4
5
2
13
6
9
12
11
8
7
10
3
0
1
4
5
6
2
15
7
10
14
13
12
9
8
11

Figure 2