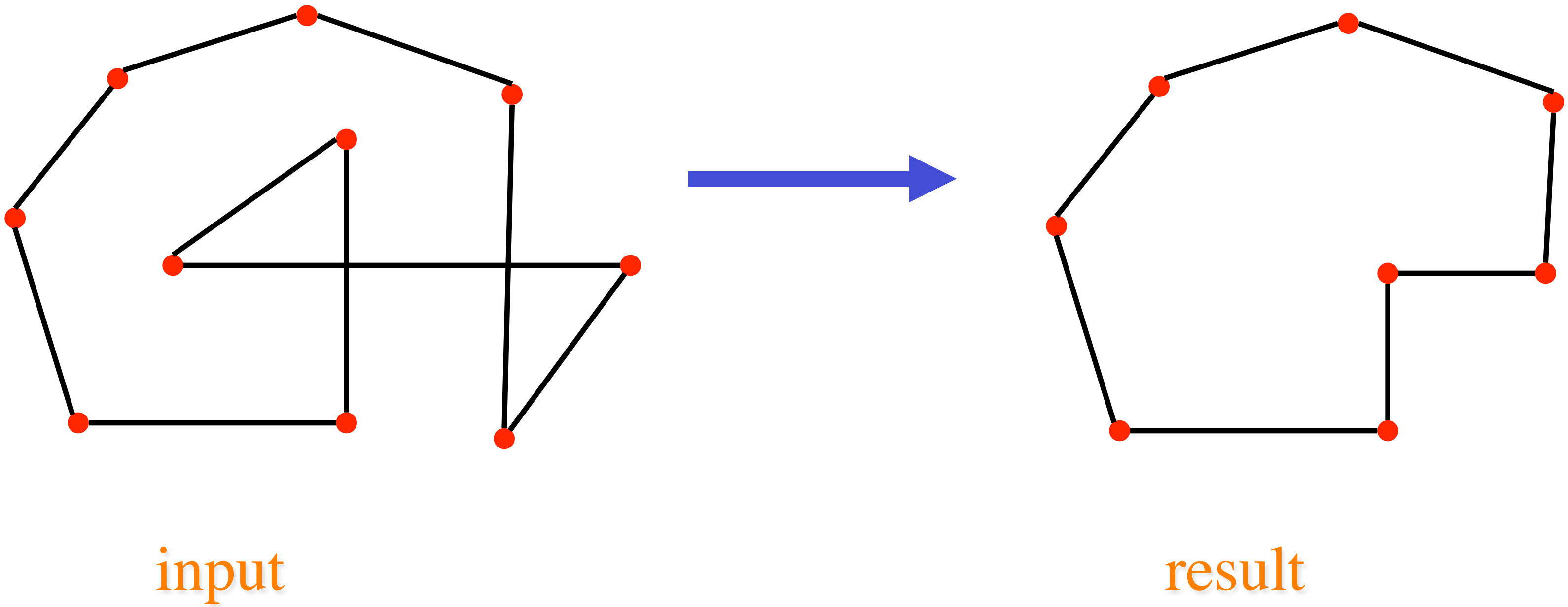
Self-trimming

- Objective:
  - We are given a polyloop that may self-intersect.
  - We want to extract from it a set of manifold curves that bound the major areas enclosed by the original curve
- Case study of how to design and implement a solution to this complex problem
  - Metaphor
  - Notation
  - Approach
  - Representations
  - Data structures
  - Debugging tools
  - Implementation
  - Documentation

Example



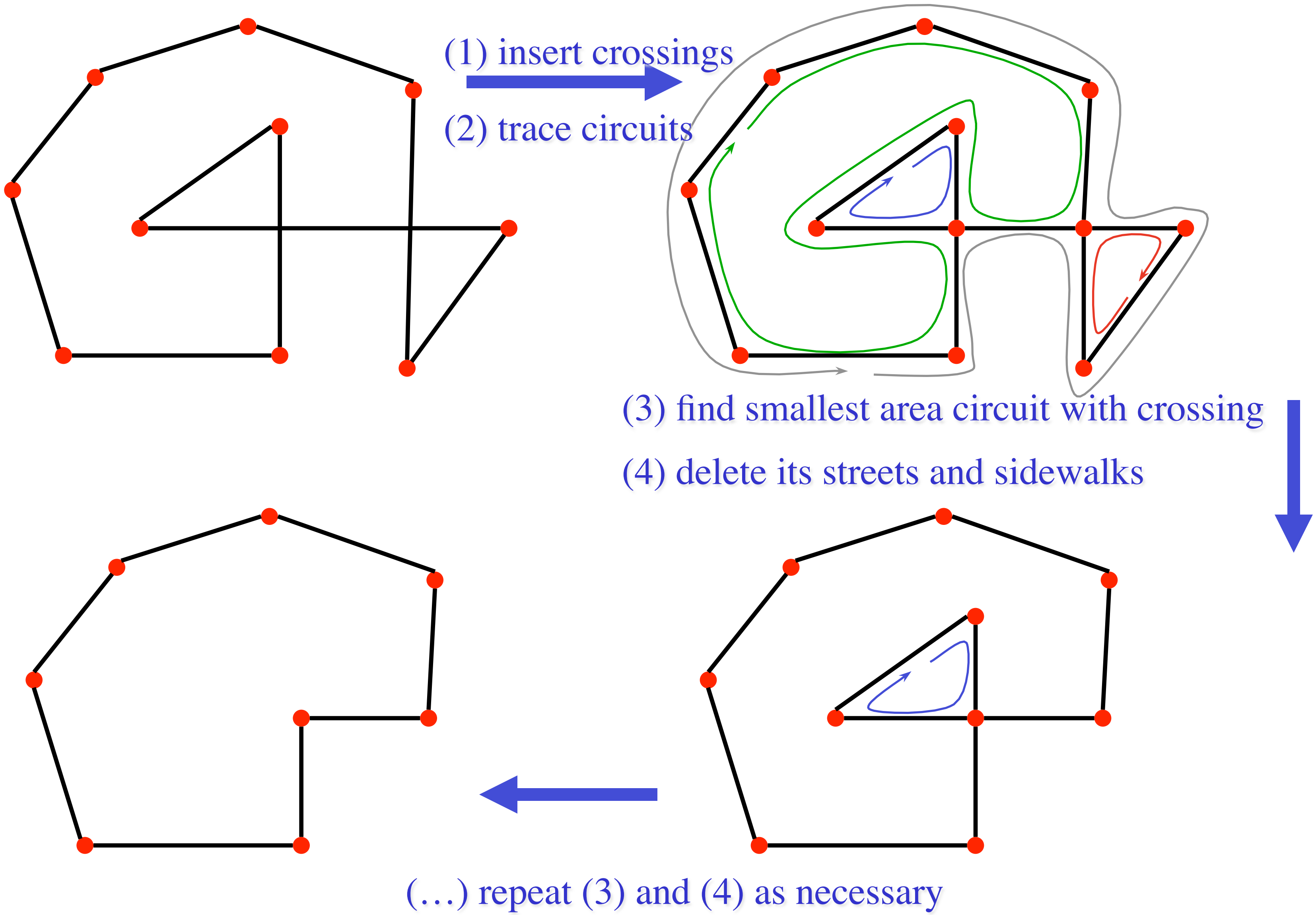
Use a metaphor to invent a solution

Invent a metaphor
- The edges are **streets**
- There are **sidewalks** along the network of streets

Reformulate your problem using your metaphor terminology
- The sidewalks form closed-loop **circuits**
- We want to identify a subset of these circuits
- If we only take the streets along these circuits, there should be no crossings

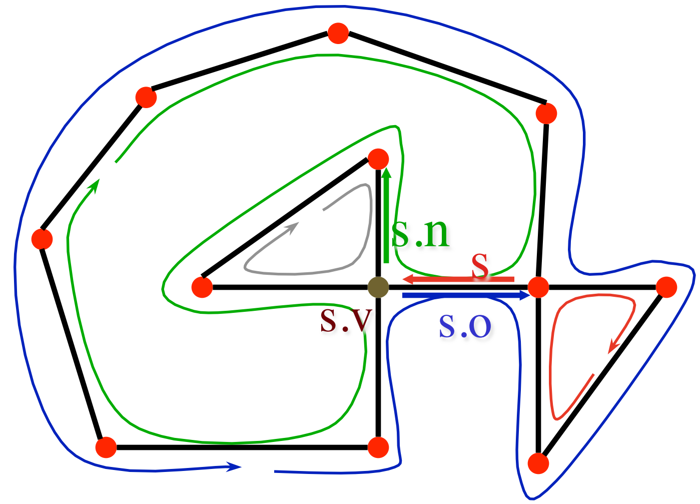
Use this terminology to **invent an approach**
- Compute all crossings and splits streets at these crossings (1)
- Identify circuits (2)
- Identify the **smallest area circuit** that goes through a **crossing** (3)
- Delete (close) its streets and sidewalks (4)
- Repeat (3) and (4) as necessary

Test example



Notation, primitive, operators

- Define your primitives
  sidewalk s (1 block strip)
- Select visualization symbol
  arrow along sidewalk
- Select operators and notation
  s.n: next sidewalk (along circuit)
  s.o: opposite sidewalk (across street)
  s.v: vertex ID of crossing reached by s
  s.g: point where s.v is located



This is a fundamental step in solving the problem.
Explore several ideas and do not hesitate to change your notation.

Initialization of sidewalks

- The input is an ordered set of vertices

  n(v) is the next vertex after v in cyclic order (use modulo)

- Initialize
- Keep track of previous step

  previous sidewalks ps, po, and previous vertex pv

- For each vertex ID v do

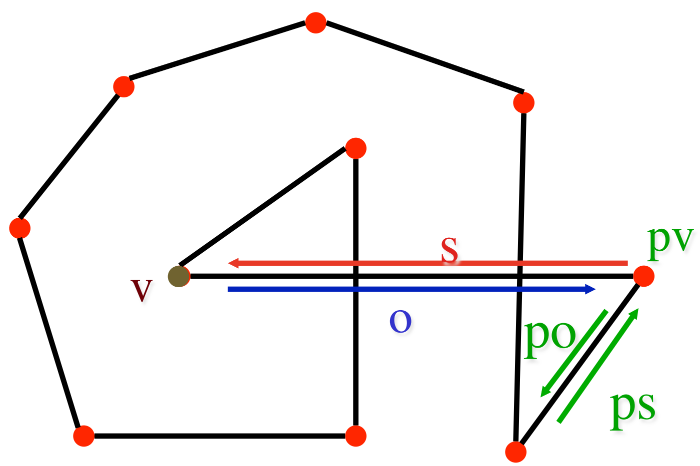
  Create two sidewalks s and o

  s.o=o; o.o=s;

  o.v=pv; s.v=v;

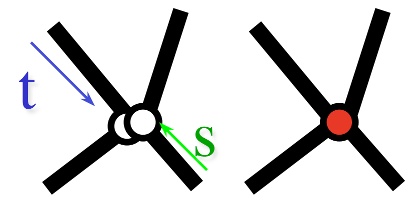
  ps.n=s; o.n=po;

  ps=s; po=o; pv=v;

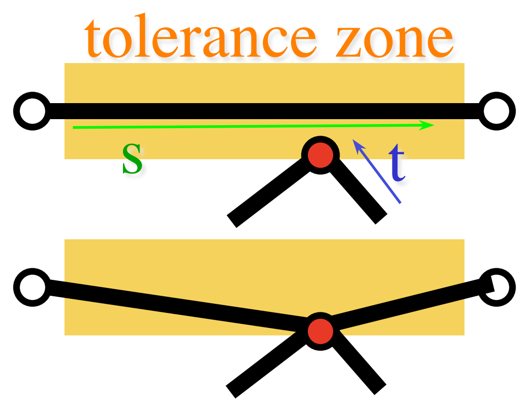


Insert crossings

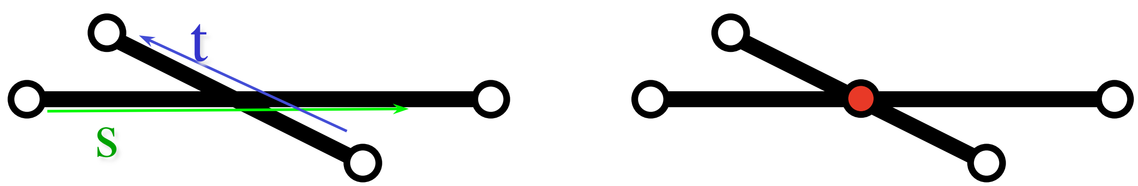
- Many cases, try to simplify the solution
  - Invent primitive steps
  - Do not worry about order yet
- Identify coincident vertices and merge them
  - mergeVertices(s,t)



- Split edge when a vertex overlaps the tolerance zone around its interior
  - insertVertex(s,t)

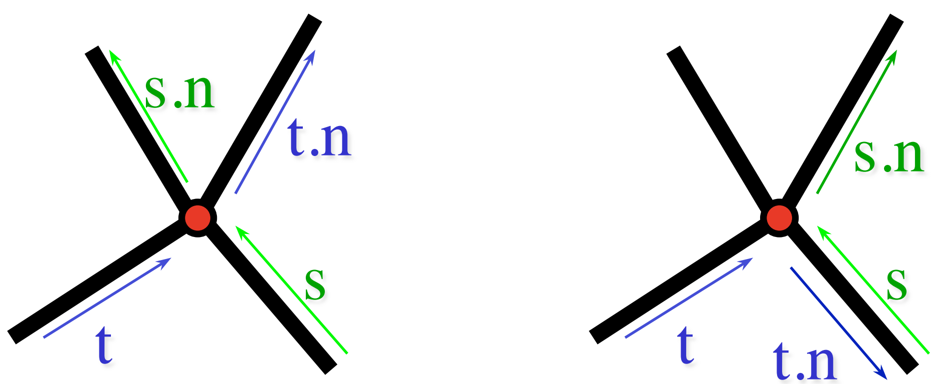


- Split edges at their intersections
  - splitEdges(s,t)



Fix the order

- For each sidewalk,
  – identify its proper next n by computing angles for all sidewalks leaving s.v
- If (n!=s.n) swap them

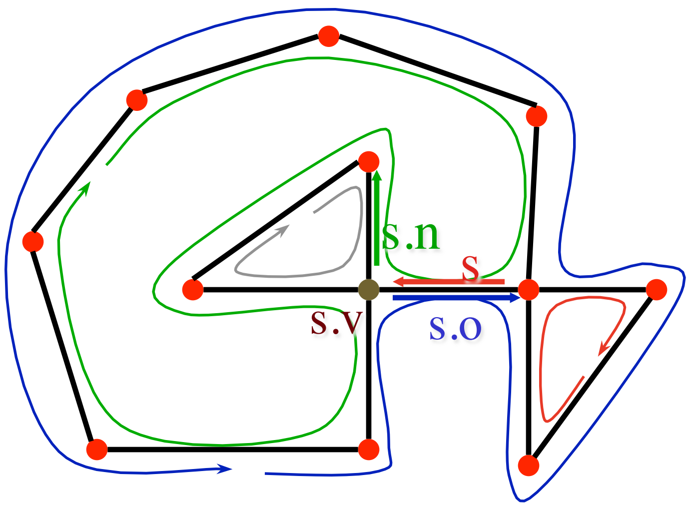


Trace the circuits

Define an algorithm at a high level

Do not commit to any particular data structure or implementation

- Associate a marker (color) with each sidewalk
- Set all markers to white
- As long as there is a white sidewalks s
- Pick a new color and trace the circuit of s
  - using t=s; repeat { … t=t.n; }
  - until you are back to s



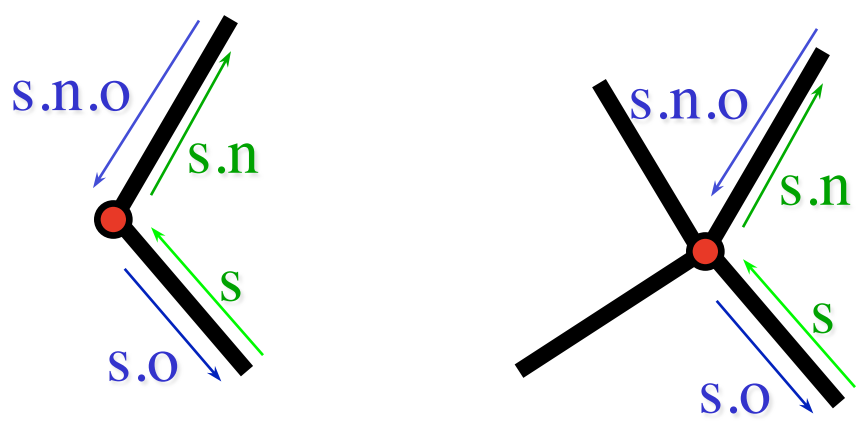
Compute the area of each circuit

- Pick a random point p
- Set area A=0;
- For each sidewalk s of circuit do

  A+=signedAreaOfTriangle(P, s.g, s.o.g);
- Return abs(A);

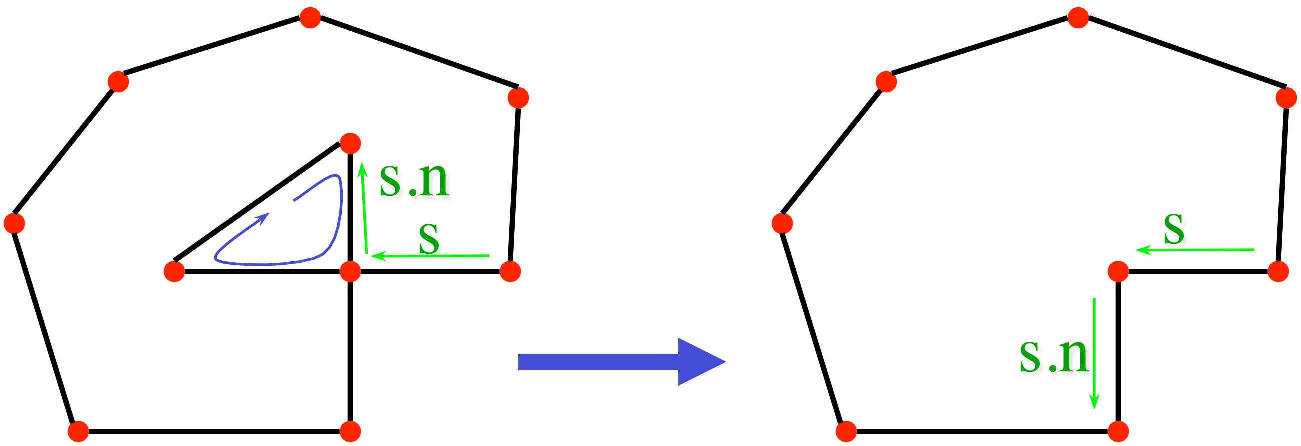
Look for crossings in circuit

if (s.n.o.n!=s.o) s leads to a crossing



Delete loop

- Fix the .n pointers of the remaining sidewalks before deleting the small loop



Data structures

- The simpler--the better
- Vertices: G[nv]; // table of points
- Sidewalks: integers in [0,ns];
- Pointers from sidewalks:

N[s] // next

O[s] // opposite

V[s] // vertex integer ID

Notation-to-code

s.n.o is O[N[s]]

s.o=t.n is O[s]=N[t]

Debugging tools

- Use graphics
- Have a global variable s
- Show s, s.n, s.o as red, green, blue arrows
- Use keys to move

  s=s.n

  s=s.o
- Check the integrity of your model
- Debug each operator
- A different key activates
  - mergeVertices
  - insertVertex
  - splitEdge

  check each one before implementing the next
- Show vertices (as dots). Crossings as larger dots.
- Display the streets of the smallest loop.
- Prepare test cases for each step.