Rensselaer Polytechnic Institute
Universidade Federal de Viçosa

Research topics in GIS

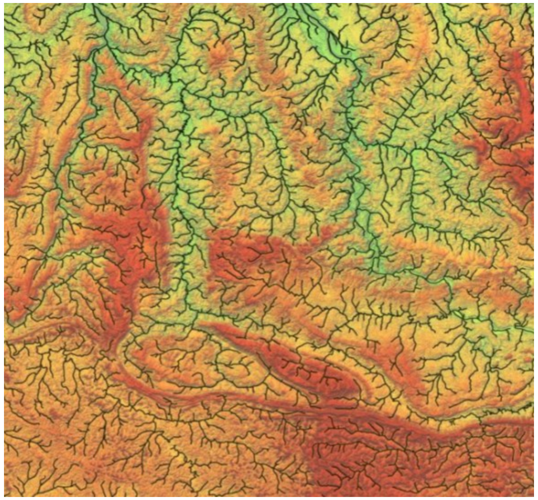
Marcus Andrade
Salles Magalhães
W. Randolph Franklin
Wenli Li

Our research

- Efficient parallel algorithms for GIS.
- Algorithms for raster and vector maps.
- Main fields in GIS:
  - Hydrography
  - Visibility
  - Operations with vector maps

Previous work: hydrography

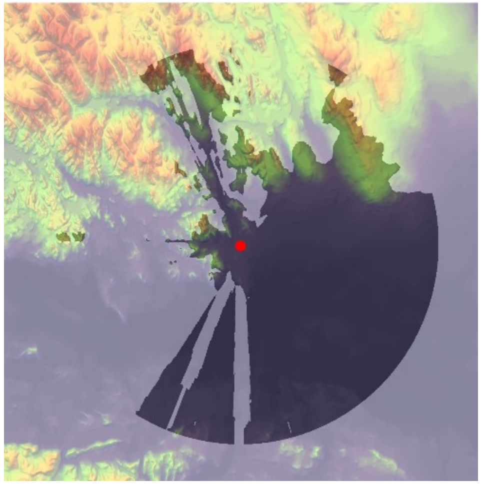
- RWFlood
  - Fast flow direction and accumulation
  - Linear-time algorithm
  - More than 100 times faster than others



- EMFlow
  - RWFlood for external memory
  - TiledMatrix (tiling+fast compression)
  - 20x faster than TerraFlow and r.watershed.seg

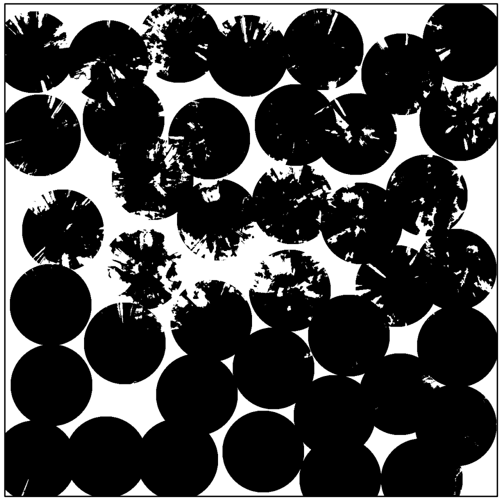
Previous work: visibility

- TiledVS
  - Visibility map computation on external memory
  - Uses TiledMatrix
- Parallel Viewshed
  - Multi-core implementation of the sweep-line viewshed
  - OpenMP
  - Up to 12x faster than the serial (using 16 threads)



Previous work: visibility

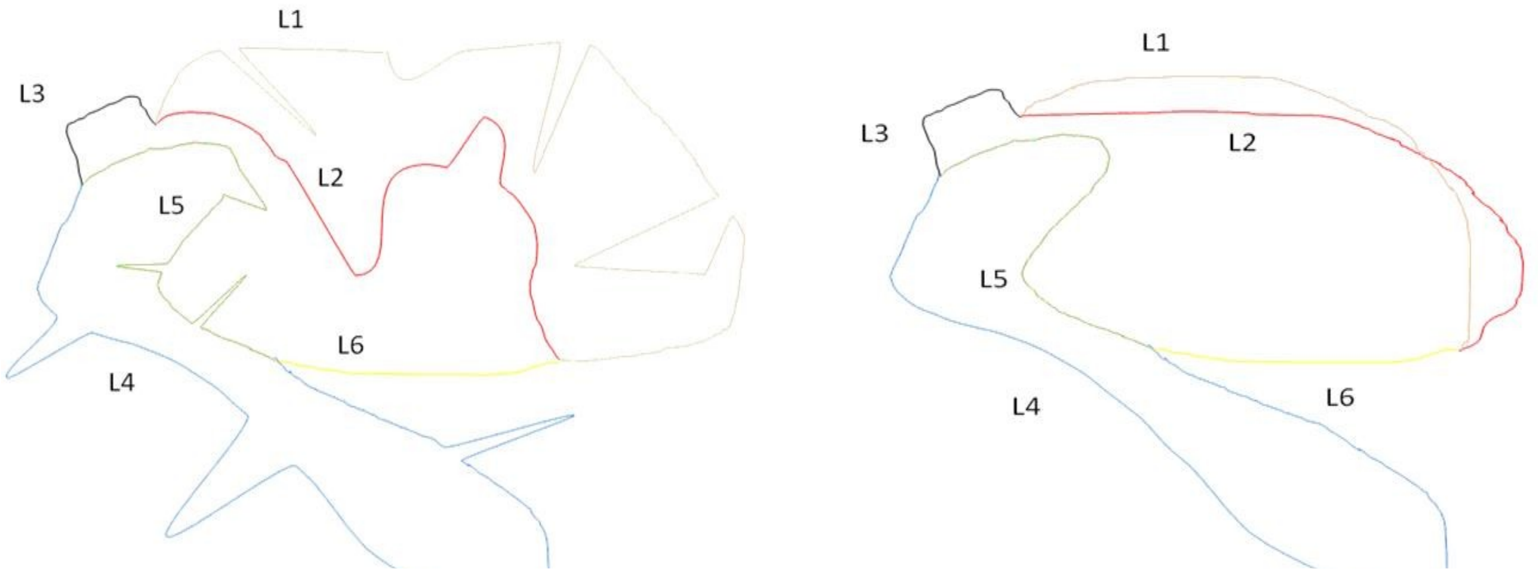
- GPU observer siting
  - Local search heuristic for observer siting
  - Given a solution S, iteractively replace S with its best neighbor
  - Neighbor(S): solution where an observer in S is replaced with an observer not in S.
  - Challenge: efficiently find the best neighbor
  - Solution: sparse matrices, adapted sparse-dense MM to compute visible areas.
    - Up to 3x faster than our previous GPU implementation.
    - Up to 7000x faster than our previous serial implementation (using dense matrices).



RPI - Rensselaer Polytechnic Institute

Previous work: map generalization

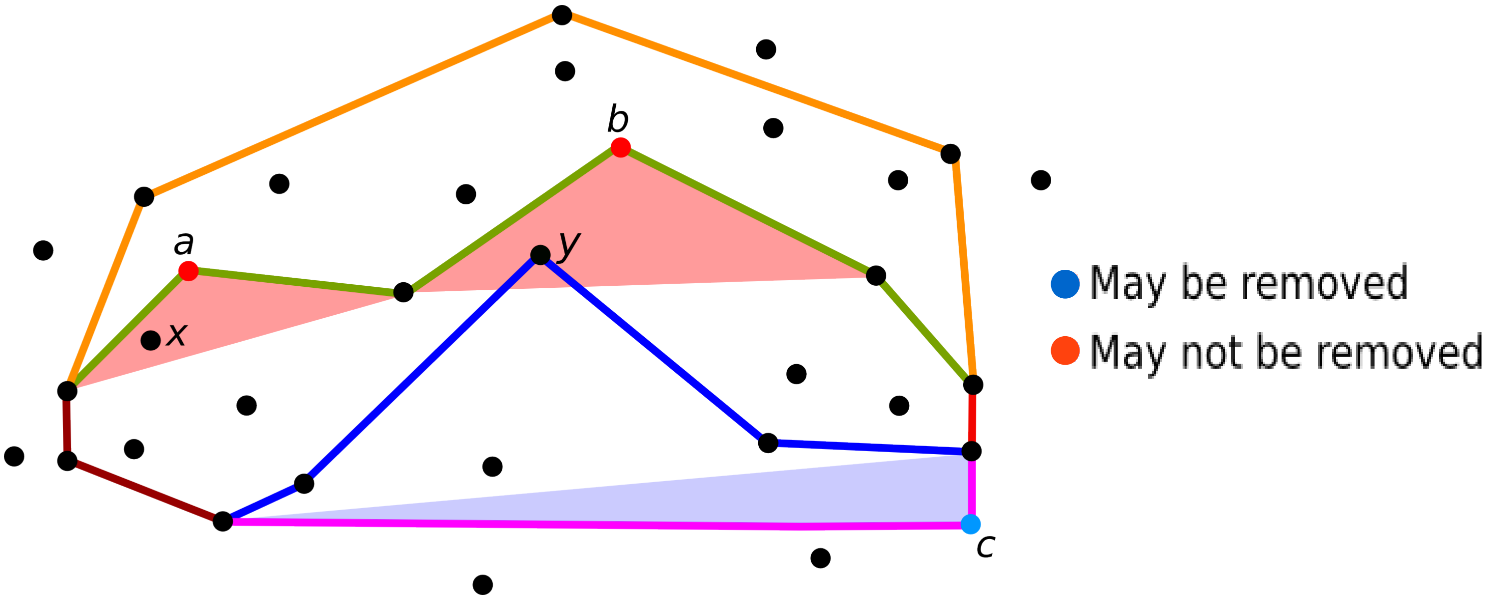
- Problem proposed in GISCUP 2014.
- Simplify polylines in a map.
- Remove points (except endpoints)
- Challenge: avoid topological problems and changes in topological relationships (control points).



Source: http://mypages.iit.edu/~xzhang22/GISCUP2014

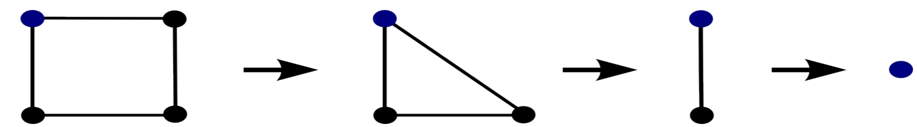
Previous work: map generalization

- Grid-Gen (ACM GISCUP)
  - Process polylines independently.
  - Remove polyline point ↔ no topological problem.
  - No topological problem ↔ no point in triangle!

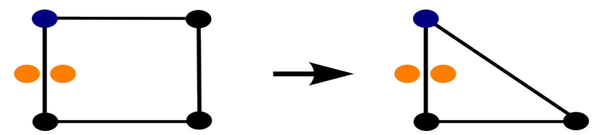


Previous work: map generalization

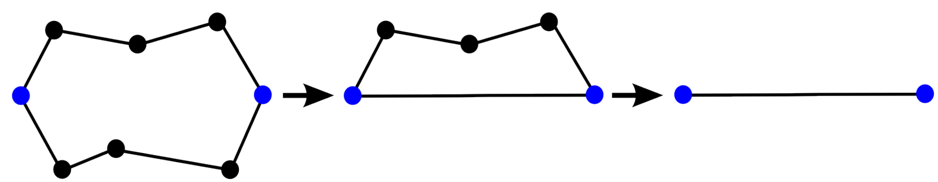
- Special cases:
  - Coincident endpoints & no control point inside.



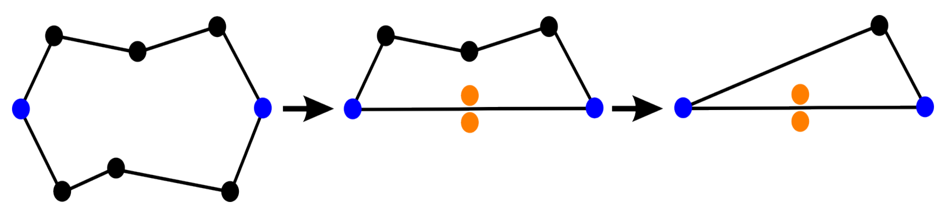
  - Solution: dummy points.



  - Two polylines with the same endpoints & no control point inside.



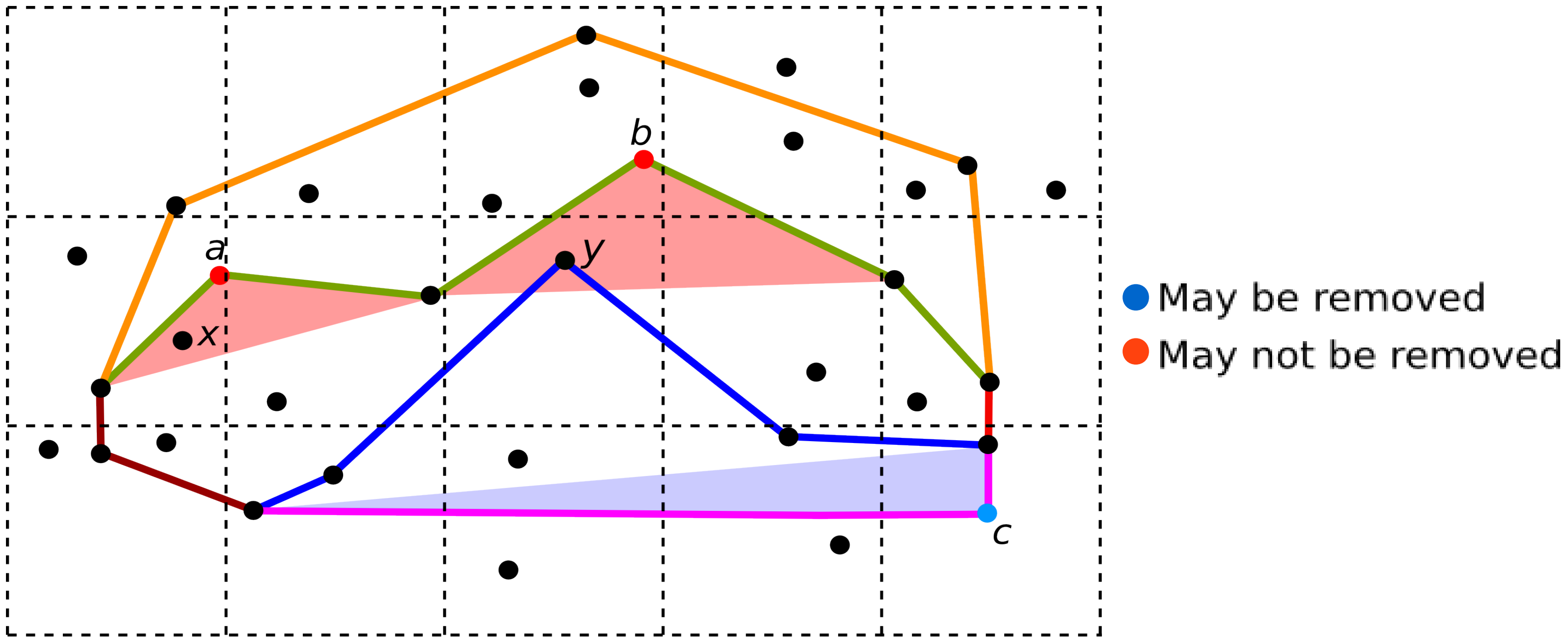
  - Also solved with dummy points.



RPI - Rensselaer Polytechnic Institute

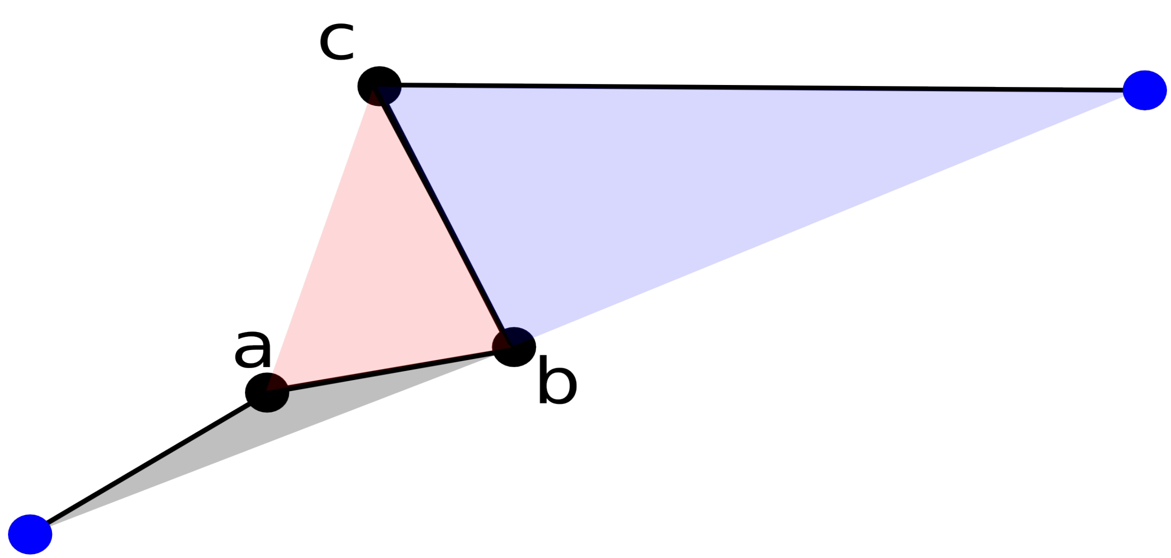
Previous work: map generalization

- For efficiency: uniform grid.
- Polylines points & control points → grid.



Previous work: map generalization

- Grid-Gen: We only try to satisfy the constraints.
- Grid-Gen2:
  - Points ranked based on “effective area” (Visvalingam-Whyatt).
  - Remove first points with small “area”.
    - Areas of neighbors are updated.
    - For efficiency → priority queue.



Previous work: map generalization

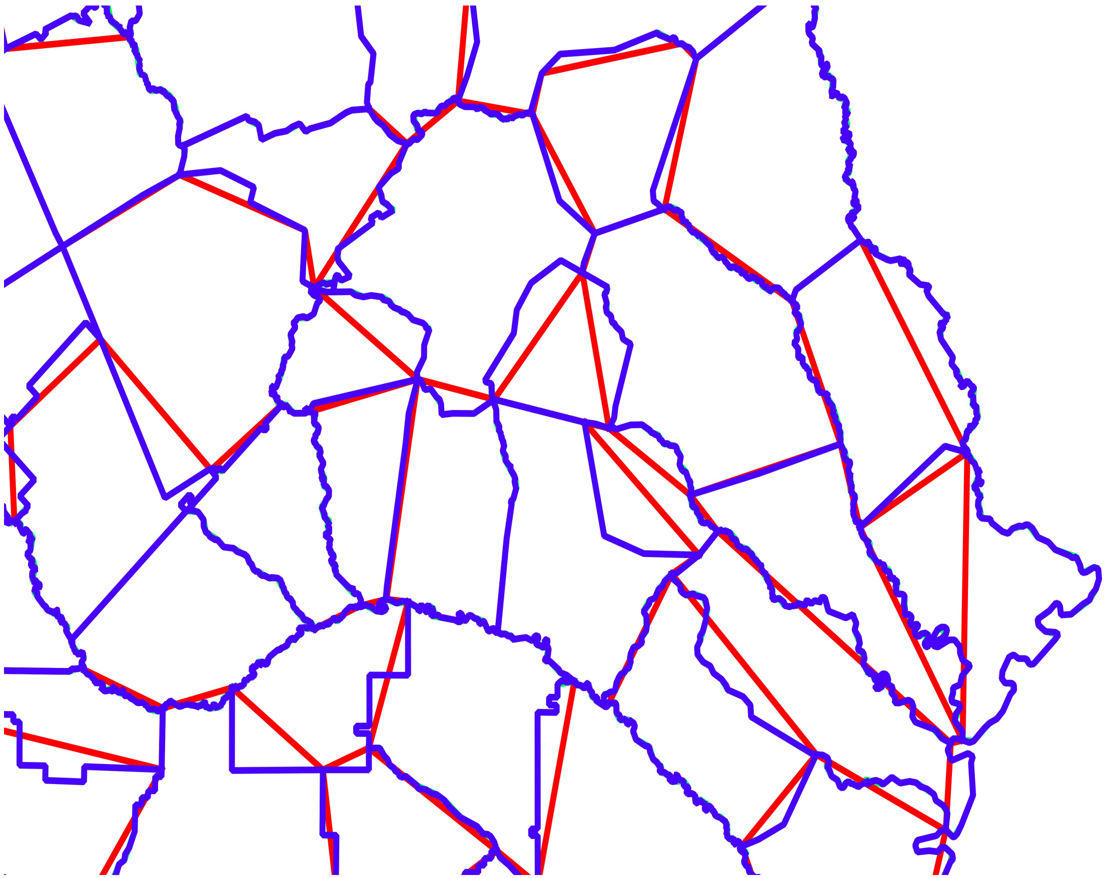
- Experiments:
- i7-3520M 3.6 GHz processor, 8GB of RAM memory
- Samsung 840 EVO SSD (500 GiB)
- Grid-Gen vs Grid-Gen2

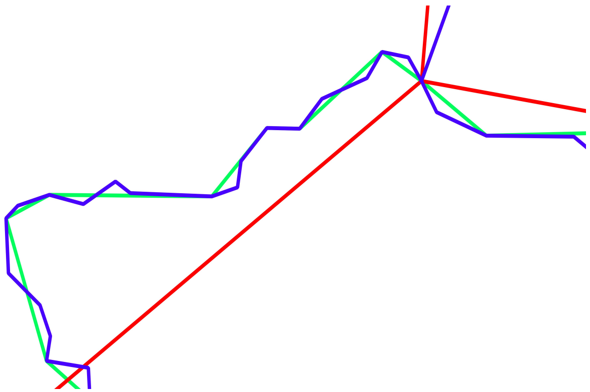
- Time (ms) for each step (only simplification is different).
- Bottleneck: **I/O** and simplification step.
- Simplification: Grid-Gen2 is 8 times slower.

Dataset # input points	3 8531	4 3×10^4	5 3×10^4	6 3×10^5	7 4×10^6
Input reading	10	22	29	257	37092
Unif. grid init.	0	1	1	24	1472
Simp. (*Grid-Gen2*)	2	15	13	435	23759
Simp. (*Grid-Gen*)	1	4	3	54	3481
Output writing	6	21	21	170	1817

Previous work: map generalization

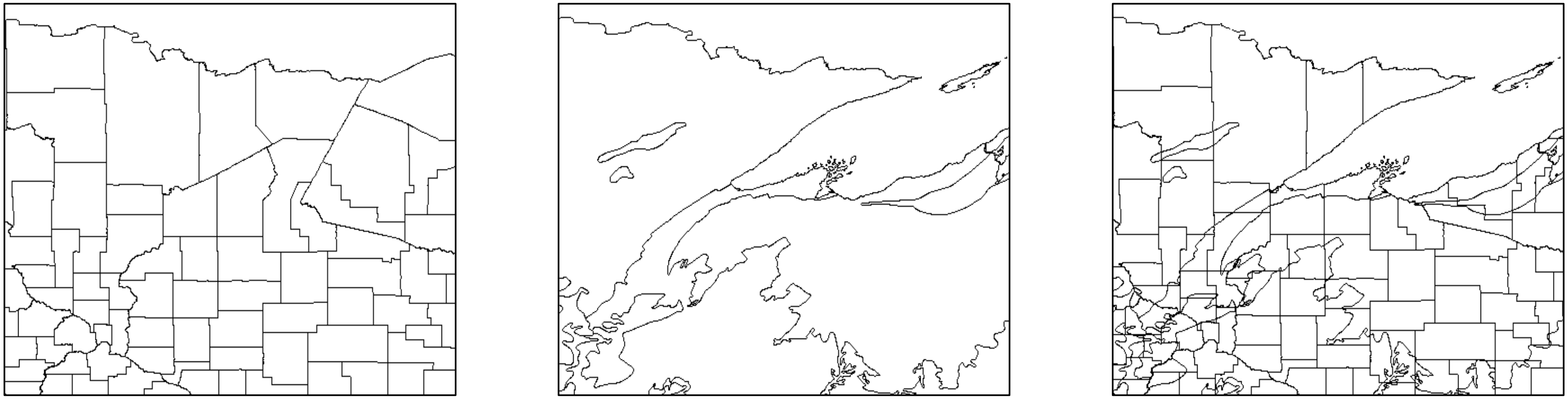
- Good visual quality:
- Example of solution (blue = original, red = Grid-Gen, green = Grid-Gen2)





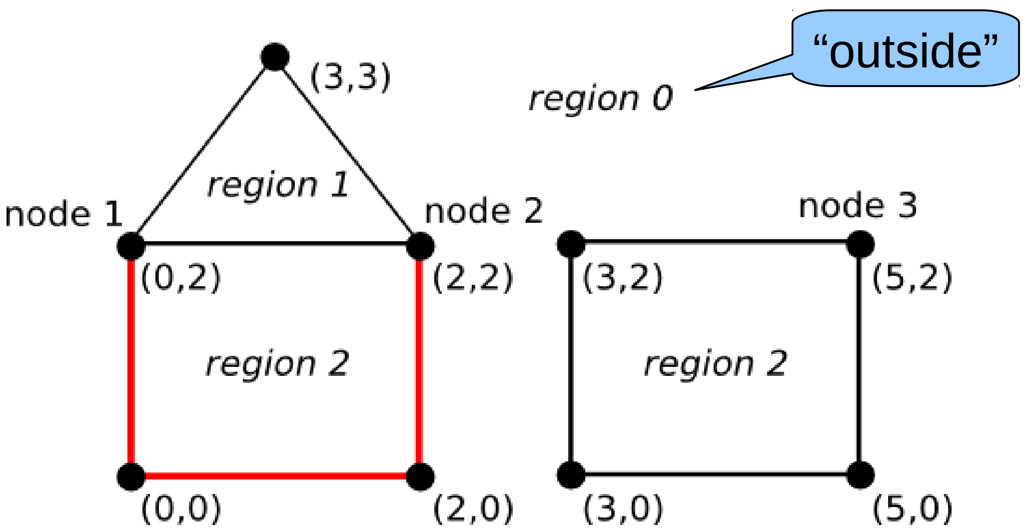
Current work: map intersection

- Finite precision of floating point → roundoff errors.
- Big amount of data → increase problem.
- Proposed solution: Rat-overlay
  - Uses rational numbers.
  - Parallelizable.



Current work: map intersection

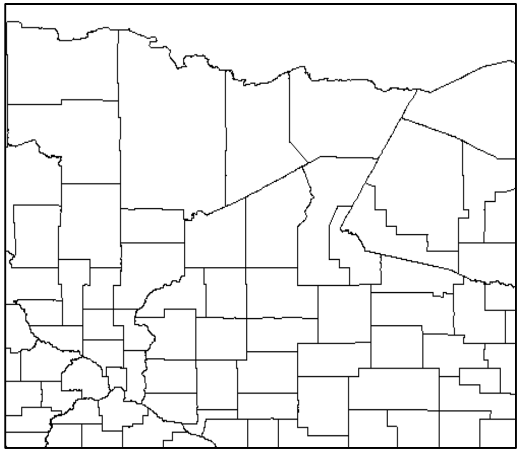
- Topological representation.
- Each region has one id.
- Edges represent boundaries.
- Sequence of edges bounding two regions:
  - chain: (id, #vertices, $node_0$, $node_1$, pol_{left}, pol_{right})

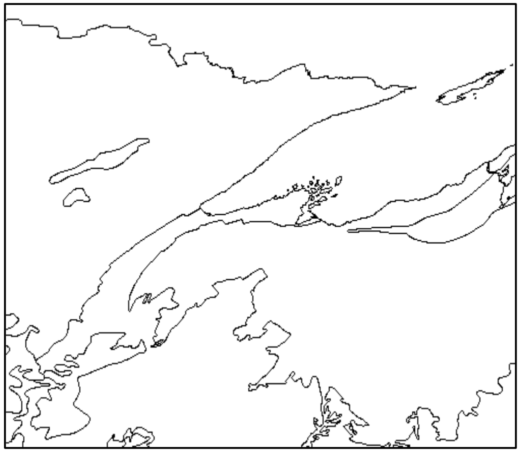


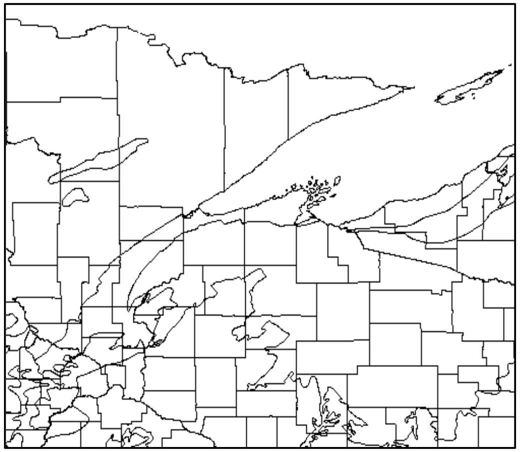
Chains:
(1,4,1,2,2,0)
(0,2);(0,0);(2,0);(2,2)
(2,2,1,2,1,2)
(0,2);(2,2)
(3,3,2,1,1,0)
(2,2);(3,3);(0,2)
(4,5,3,3,2,0)
(5,2);(3,2);(3,0);(5,0);(5,2)

Current work: map intersection

- Algorithm:
  - Find all intersections.
  - Locate vertices in the other map.
  - Compute output polygons.

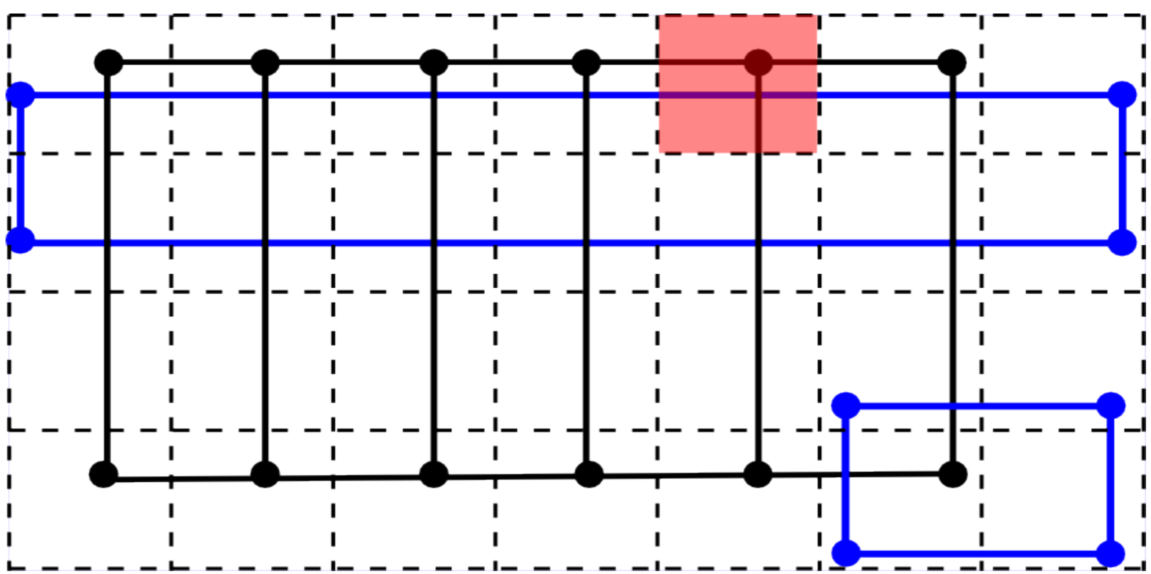






Current work: map intersection

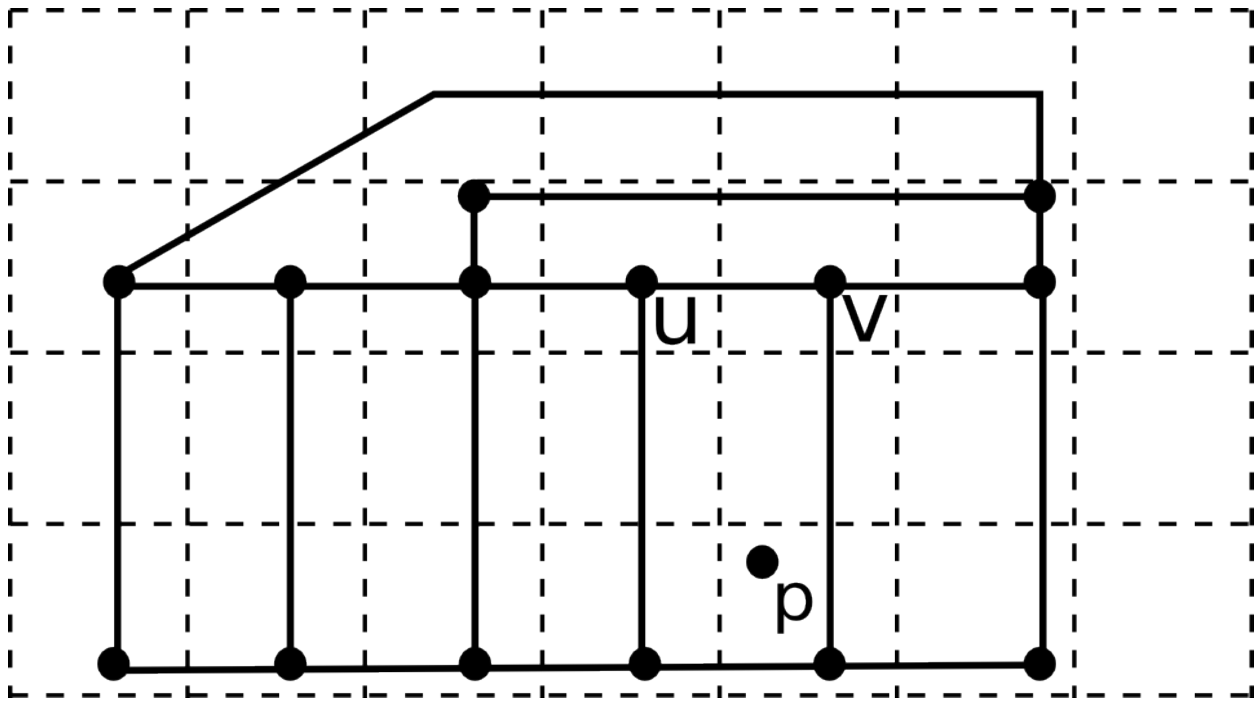
- Computing the intersections
- Test pair of edges for intersection.
- For efficiency: uniform grid.
  - Insert edges in grid cells (edge may be in several cells).
  - For each grid cell c, compute intersections in c.



4x7 uniform grid.
Blue map: 8 edges
Black map: 16 edges

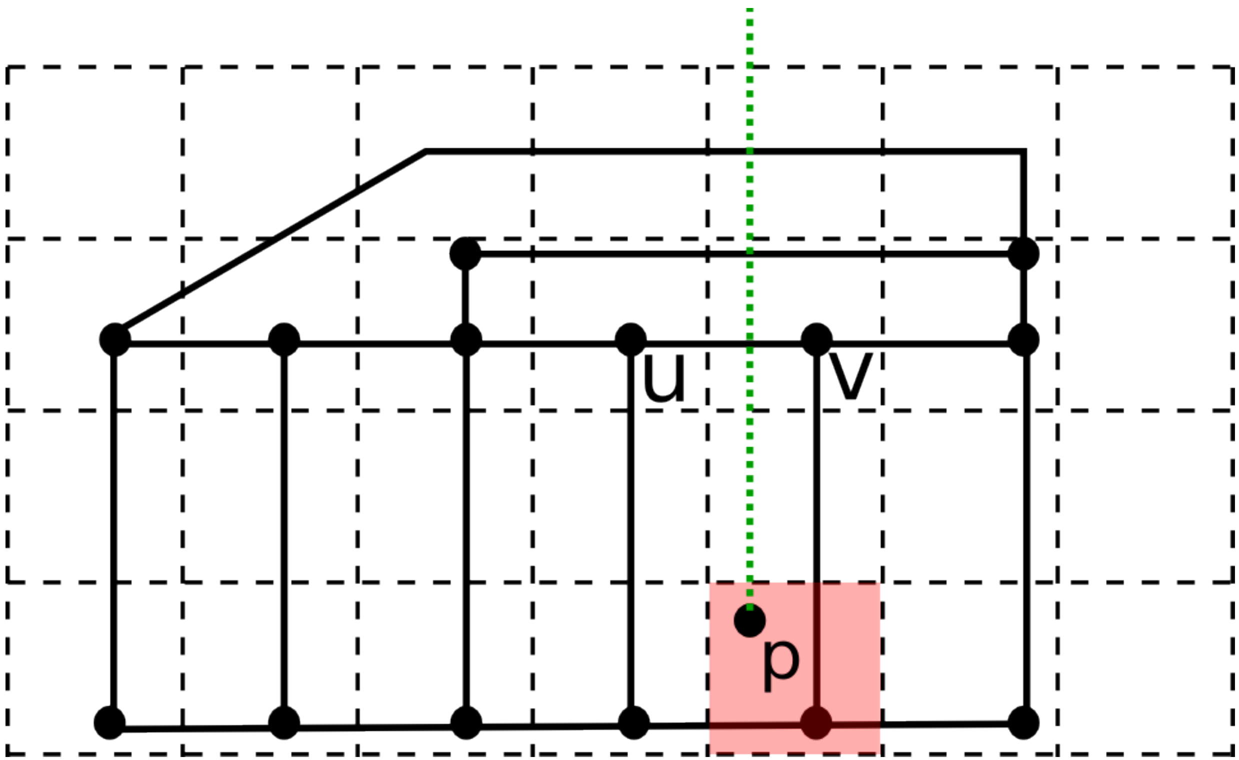
Current work: map intersection

- Locating vertices in the other map
- Also implemented using a uniform grid.
- Given *p*, find the lowest edge above *p*.



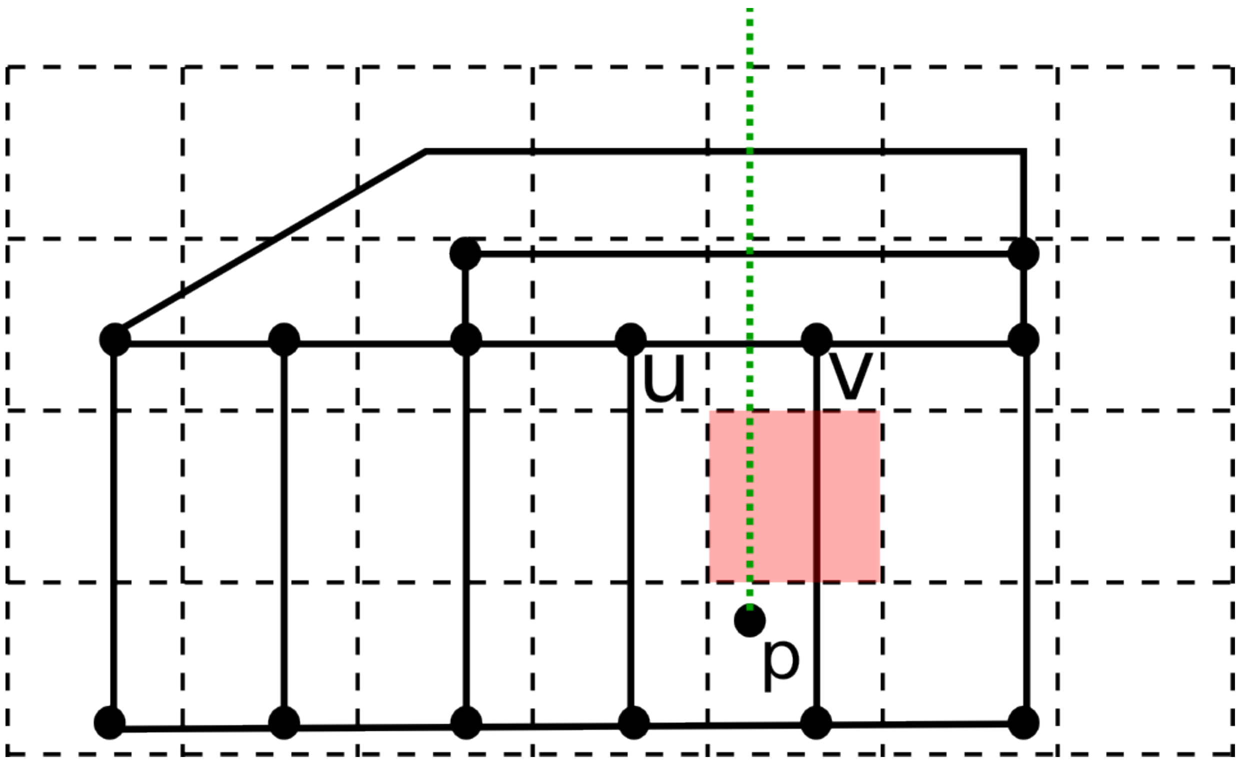
Current work: map intersection

- Locating vertices in the other map
- Also implemented using a uniform grid.
- Given *p*, find the lowest edge above *p*.



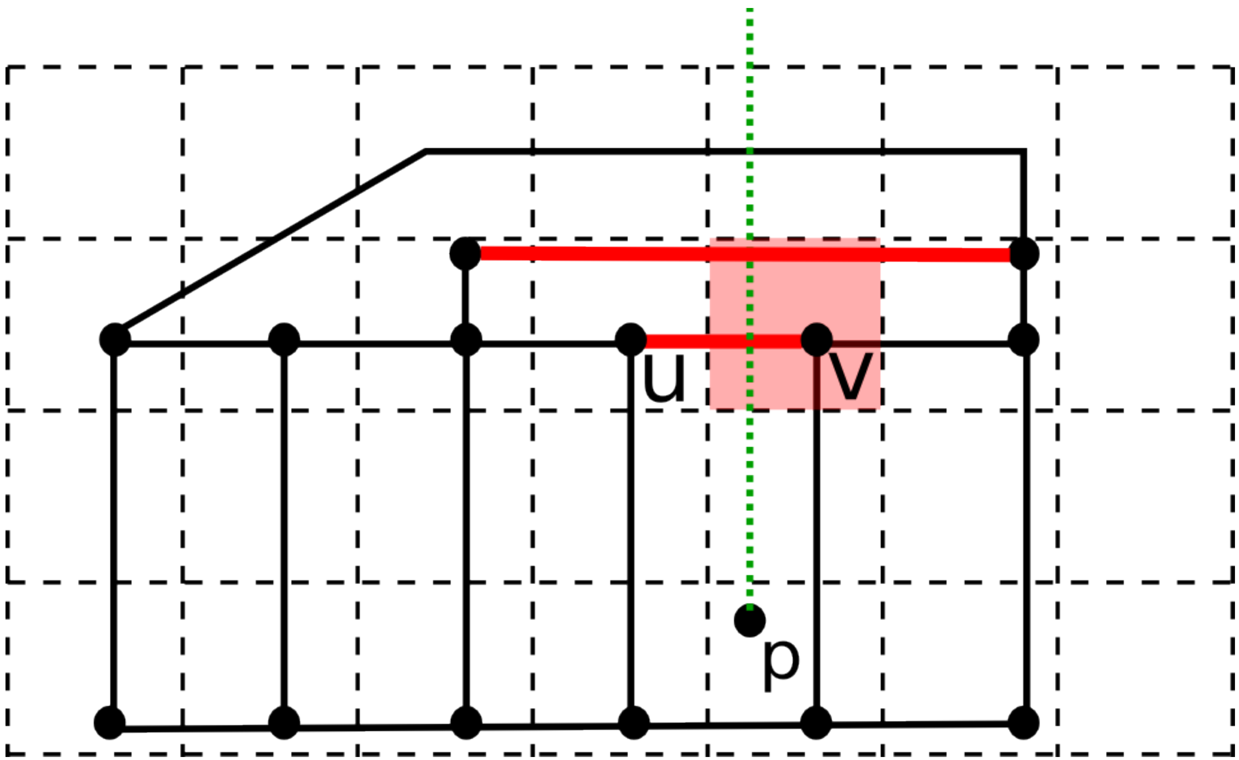
Current work: map intersection

- Locating vertices in the other map
- Also implemented using a uniform grid.
- Given *p*, find the lowest edge above *p*.



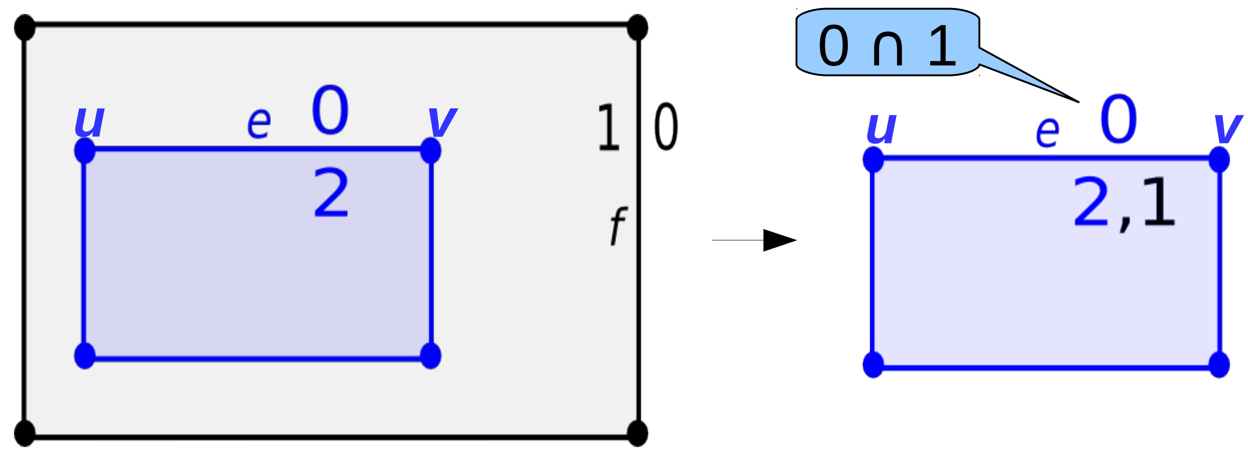
Current work: map intersection

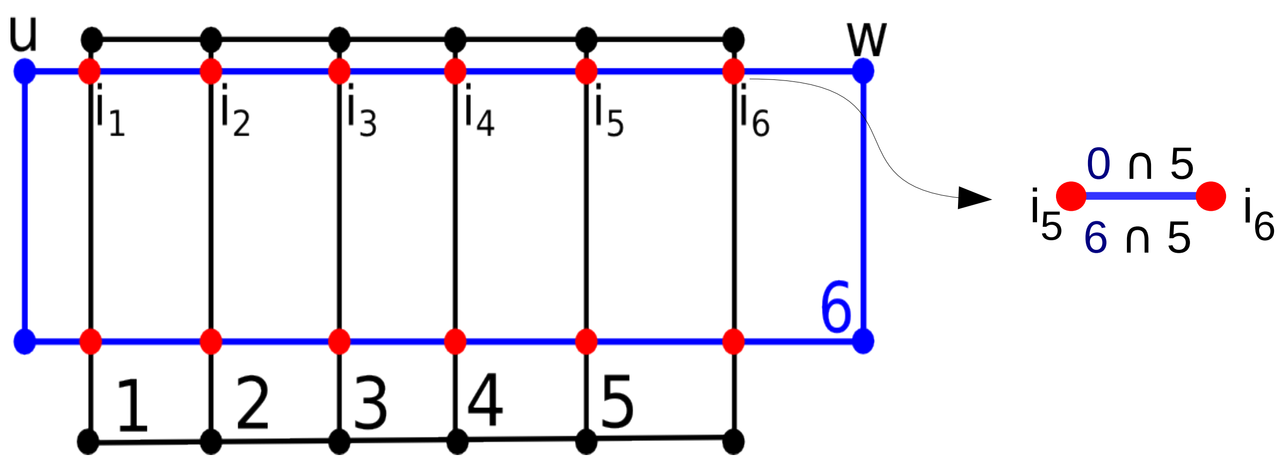
- Locating vertices in the other map
- Also implemented using a uniform grid.
- Given *p*, find the lowest edge above *p*.



Current work: map intersection

- Finally: edges are classified

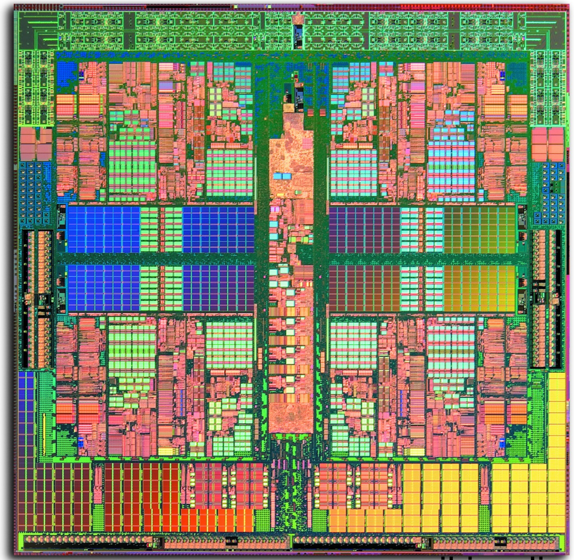




Current work: map intersection

- This algorithm → few data dependency → very parallelizable.
  - Uniform grid creation: edges in parallel.
  - Locate vertices in polygons.
  - Compute intersections: cells in parallel.
  - Compute output edges: process input edges in parallel.

- Implemented using C++/OpenMP.



source: wikipedia

Current work: map intersection

- Computation is performed using rational numbers → no roundoff errors.
- Rat-overlay implemented using GMPXX.
- Special cases: simulation of simplicity.

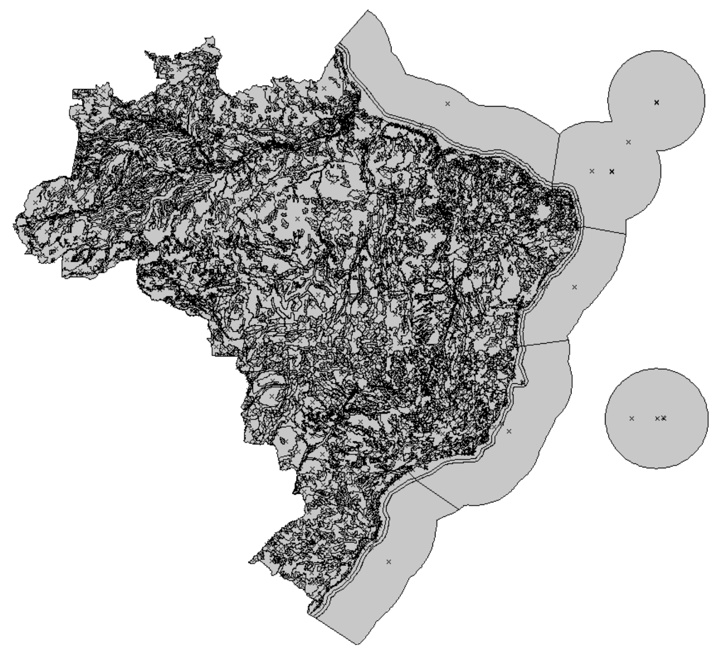
Current work: map intersection

- Rat-overlay implemented in C++ .
- Tests:
  - Dual Xeon E5-2687 → 16 cores / 32 threads.
  - 128 GiB of RAM.
  - Linux Mint 17

Current work: map intersection

- 2 Brazilian and 2 North American datasets.
- Shapefiles converted to our format.

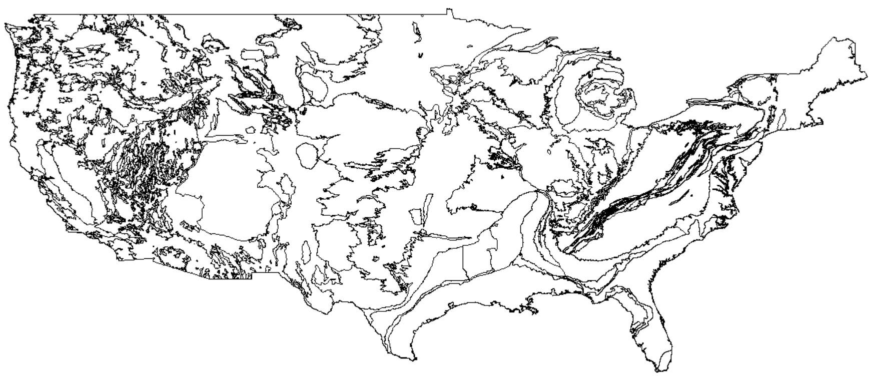
- BrCounty: 342,738 vertices, 2,959 polygons
- BrSoil: 258,961 vertices, 5,567 polygons.

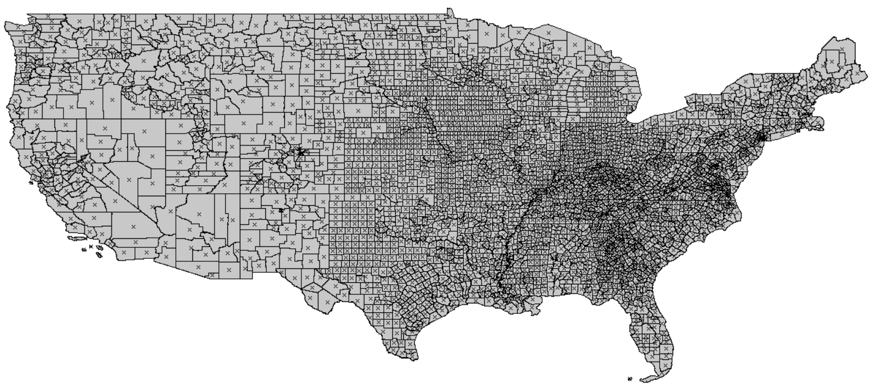


Current work: map intersection

- 2 Brazilian and 2 North American datasets.
- Shapefiles converted to our format.

- UsAquifers: 195,276 vertices, 3,552 polygons
- UsCounty: 3,648,726 vertices, 3,110 polygons





Current work: map intersection

- Sequential vs Parallel Rat-overlay vs GRASS GIS (sequential).
- Parallel:
  - Always faster than GRASS.
  - Speedup << 32
    - Critical sections.
    - 16 physical cores.
    - Amdahl's law.

Map 1	Map 2	# intersections	Grid size	Time (s)		
				Serial	Parallel	GRASS
BrCounty	BrCounty	105,754	2,000	34.5	11.5	30.3
BrSoil	BrSoil	56,246	2,000	23.3	7.4	32.3
BrCounty	BrSoil	20,860	1,000	16.1	5.9	81.7
UsAquifers	UsAquifers	50,329	8,000	37.2	11.9	47.3
UsCounty	UsCounty	300,511	16,000	625.5	124.4	175.0
UsCounty	UsAquifers	11,744	8,000	67.5	28.3	86.3

Current work: map intersection

- Time (secs.) spent in each step.
- We used the best grid size.
- I/O: 16% to 38% of time.
- Edge intersection time: big mainly when intersecting same map.

Map 1 Map 2	BrCounty BrCounty	BrSoil BrSoil	BrCounty BrSoil	UsAquifers UsAquifers	UsCounty UsAquifers	UsCounty UsCounty
I/O	2.4	1.6	1.9	2.2	10.9	20.4
Compute areas	0.5	0.3	0.2	0.3	1.1	3.1
Create grid	1.7	1.3	1.1	3.5	7.4	17.7
Intersect edges	2.3	1.7	0.7	3.0	2.0	60.6
Locate points	1.6	0.8	0.9	1.6	4.7	13.7
Compute output	3.0	1.6	1.0	1.3	2.3	9.0
Total	11.5	7.4	5.9	11.9	28.3	124.4

Current work: map intersection

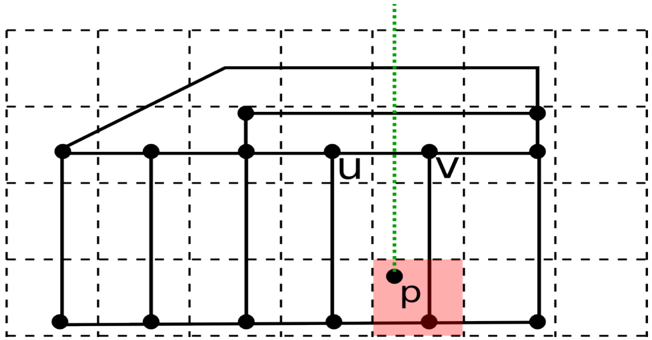
- Bottleneck: Edge-edge intersections.
- We've been trying to improve this step.
  - Problem: parallel memory allocation when rational numbers are created.
  - Solution: avoid creating “local” temporary rationals.
  - The new version:
    - 17 seconds (vs 60 seconds) for intersecting US_County with itself.
    - More scalable: 16 times speedup (vs 8x) if compared with the serial version.

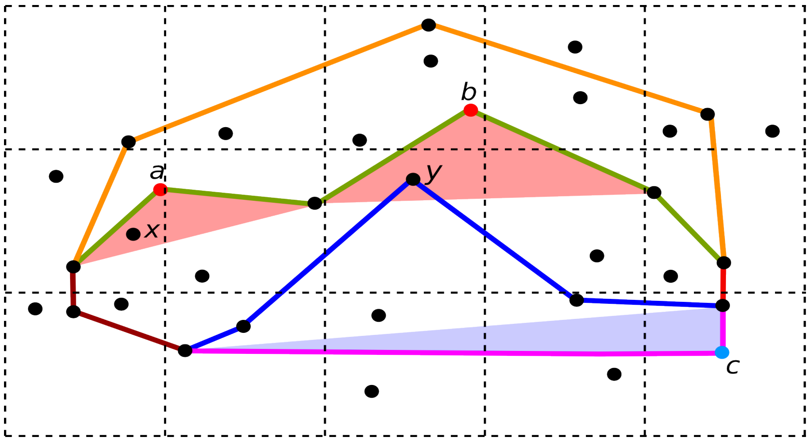
Future work

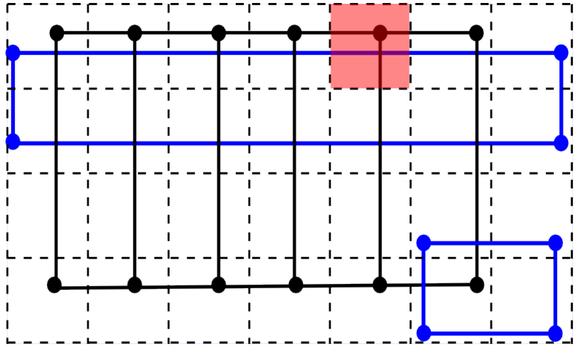
- Automatic map cleanup.
  - GIS such as GRASS have some cleanup tools.
  - Not well documented.
  - Frequently do not work very well.
  - Our idea: develop automatic map cleanup tools.
  - Useful for the intersection problem.

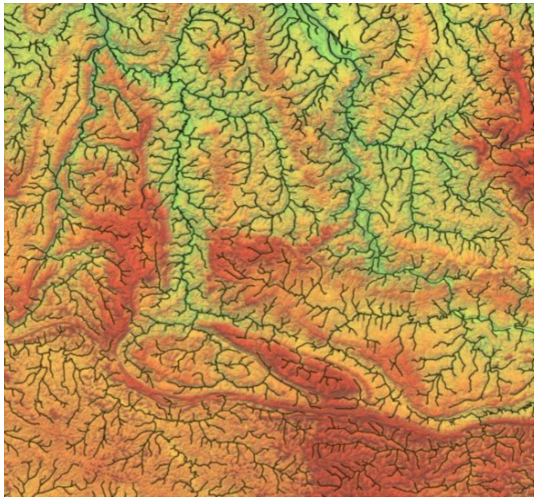
- Intersection in 3D.
  - Perform exact 3D intersection.
  - Use rationals.

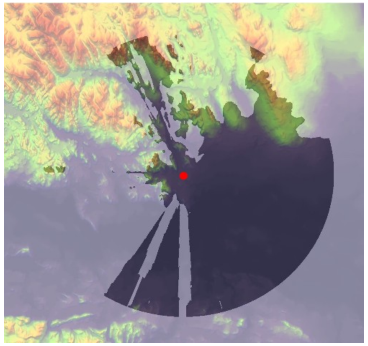
Any questions or suggestions?











Acknowledgement:

Salles Magalhaes
vianas@rpi.edu