Rensselaer Polytechnic Institute
Universidade Federal de Viçosa

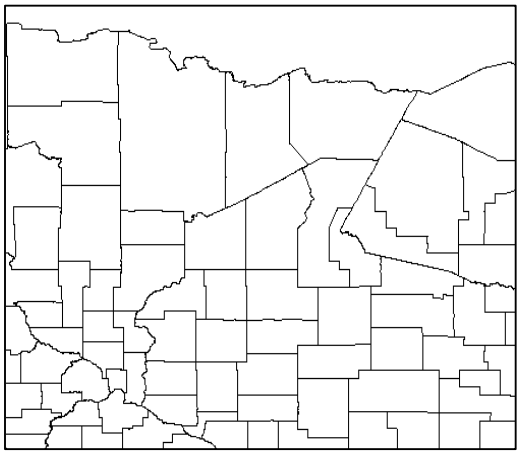
Exactly computing map overlays using rational numbers

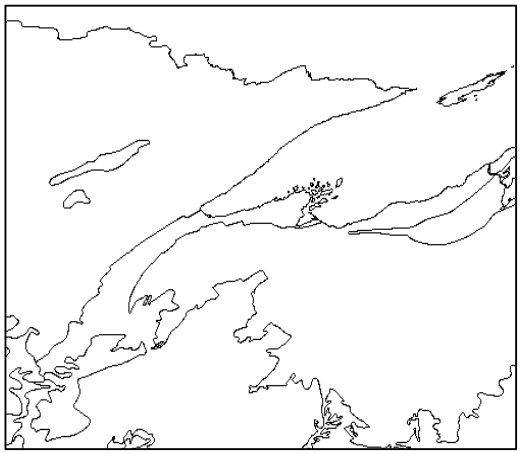
W. Randolph Franklin
Salles Viana Gomes de Magalhães

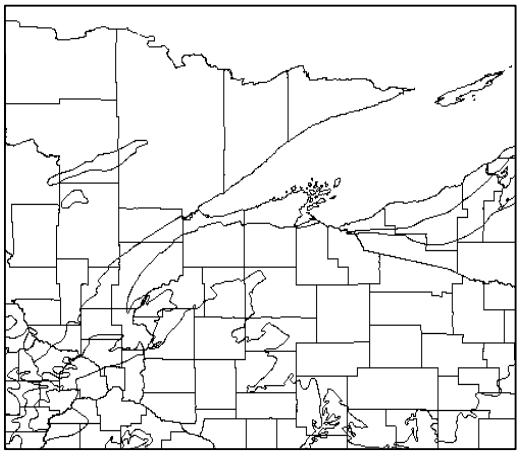
AutoCarto 2014
Pittsburgh, Pennsylvania

Map overlay

- Two vectorial maps are superimposed.
- The intersection between polygons from the two maps is computed.
- Several applications. Ex: counties and watersheds.







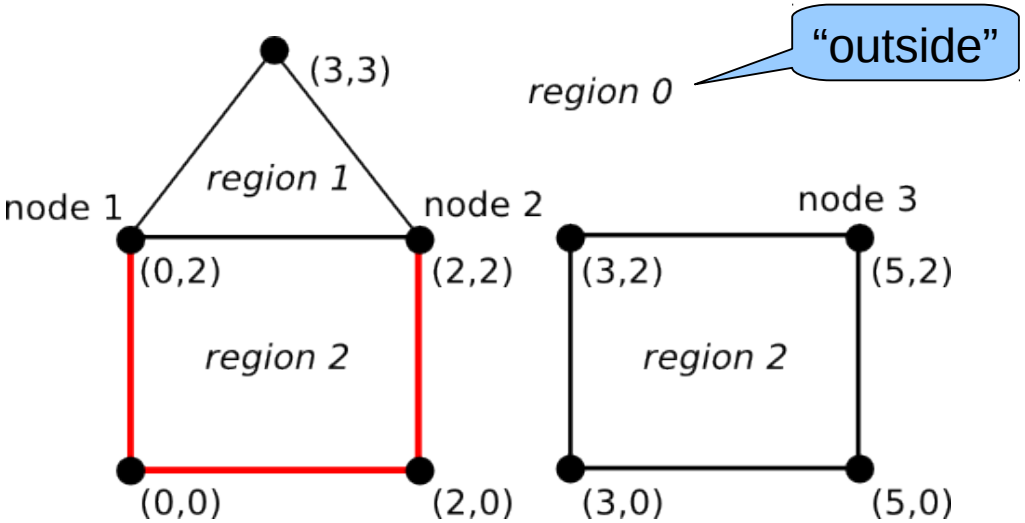
Challenge

- Finite precision of floating point → roundoff errors.
- Big amount of data → increase problem.

- Proposed solution: Rat-overlay
  - Uses rational numbers.
  - Parallelizable.

Map representation

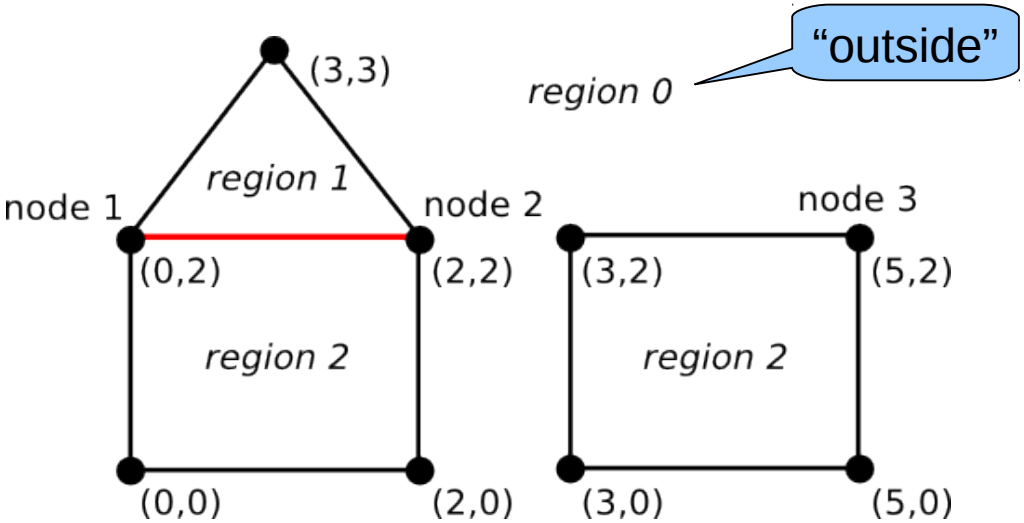
- Topological representation.
- Each region has one id.
- Edges represent boundaries.
- Sequence of edges bounding two regions:
  - chain: (id, #vertices, $node_0$, $node_1$, pol_{left}, pol_{right})



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)

Map representation

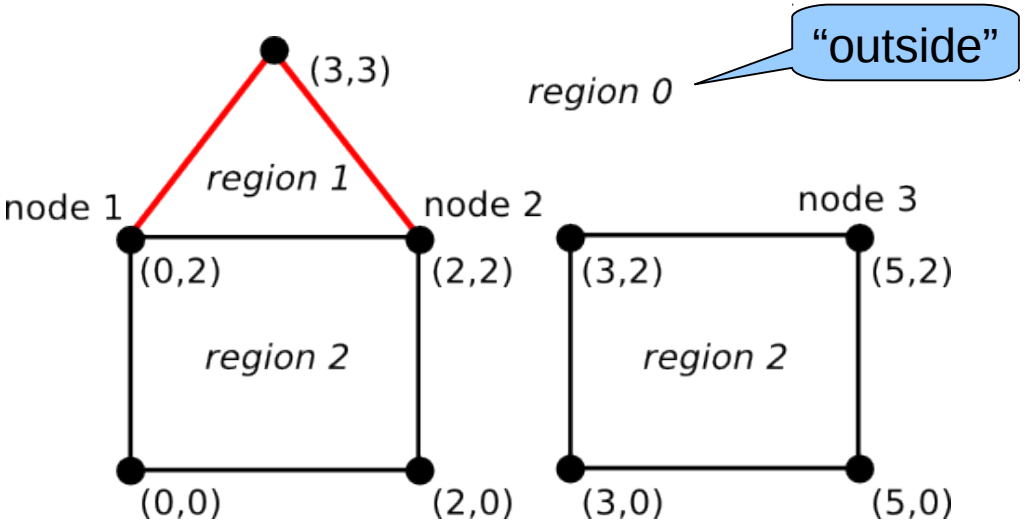
- Topological representation.
- Each region has one id.
- Edges represent boundaries.
- Sequence of edges bounding two regions:
  - chain: (id, #vertices, $node_0$, $node_1$, pol_{left}, pol_{right})



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)

Map representation

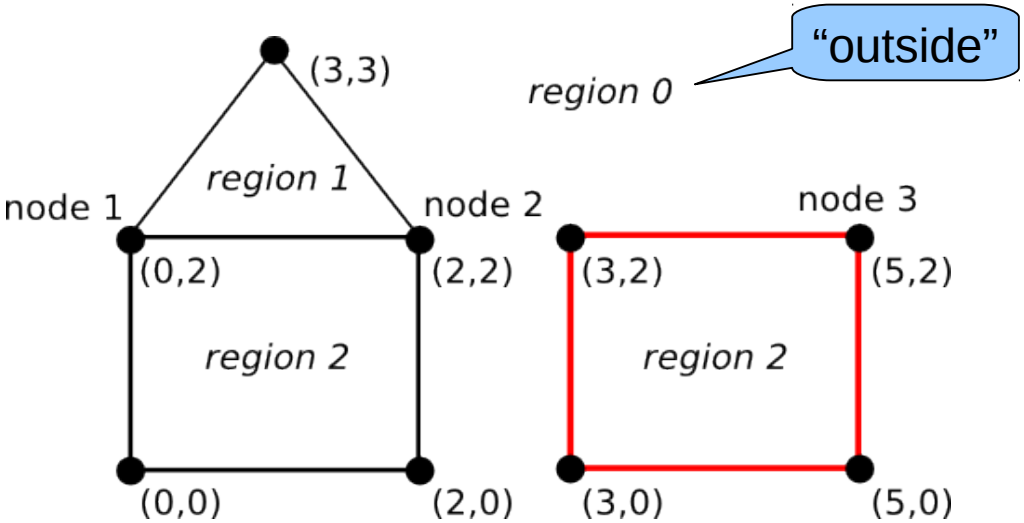
- Topological representation.
- Each region has one id.
- Edges represent boundaries.
- Sequence of edges bounding two regions:
  - chain: (id, #vertices, $node_0$, $node_1$, pol_{left}, pol_{right})



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)

Map representation

- 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})



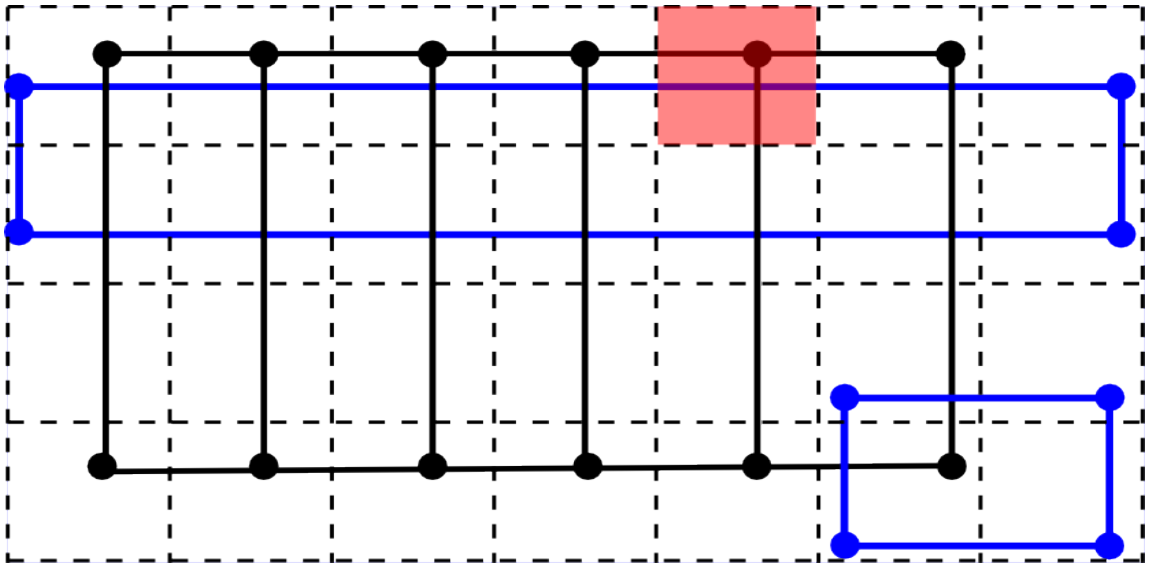
Chains:
(1,4,1,2,2,0)
(0,2);(0,0);(2,0);(2,2)
(2,2,1,2,1,2)
(0,2);(2,2)
(3,3,2,1,1,0)
(2,2);(3,3);(0,2)
(4,5,3,3,2,0)
(5,2);(3,2);(3,0);(5,0);(5,2)

Overlay algorithm

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

Finding intersections

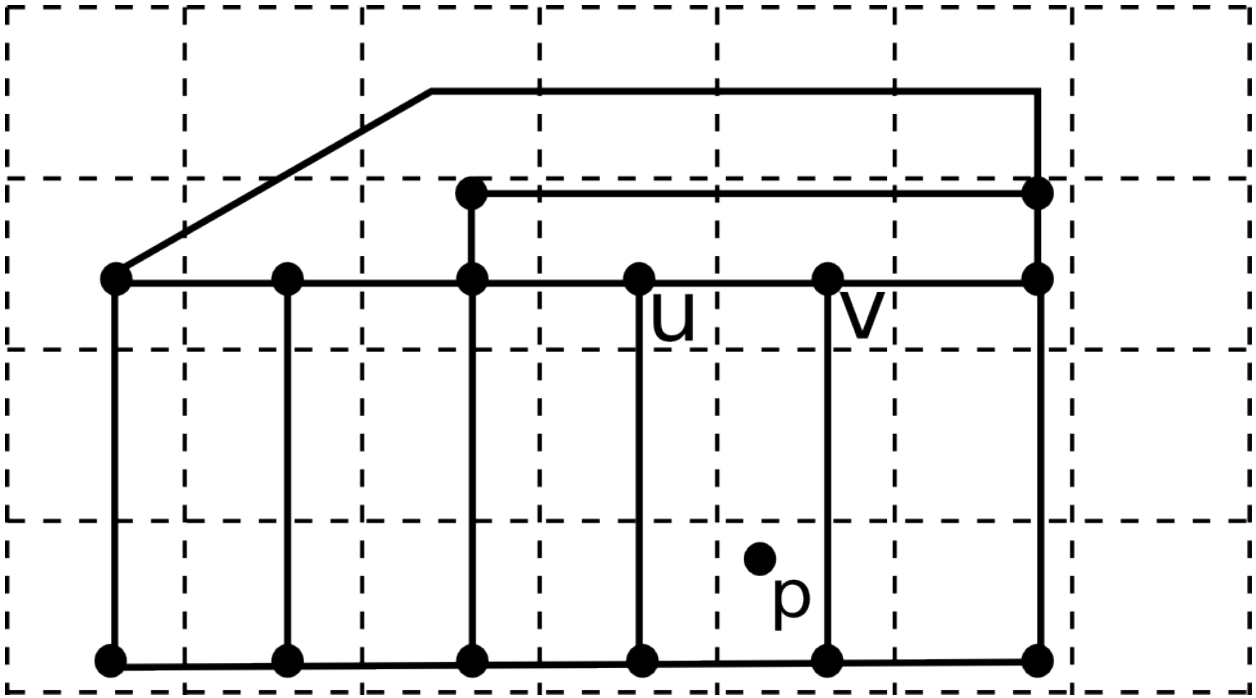
- Test pair of edges for intersection.
- For efficiency: uniform grid.
  - Insert edges in grid cells (edge may be in several cells).
  - For each grid cell c, compute intersections in c.



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

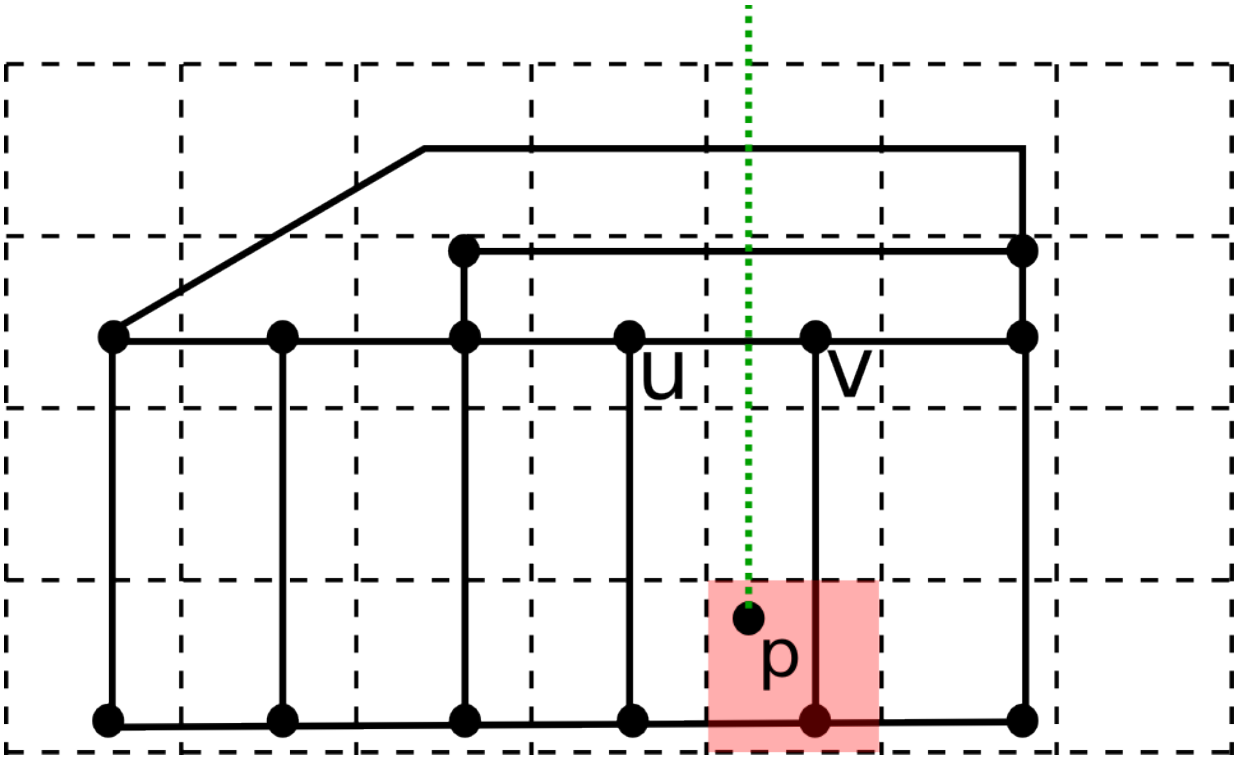
Locating vertices in other map

- Also implemented using a uniform grid.
- Given *p*, find the lowest edge above *p*.



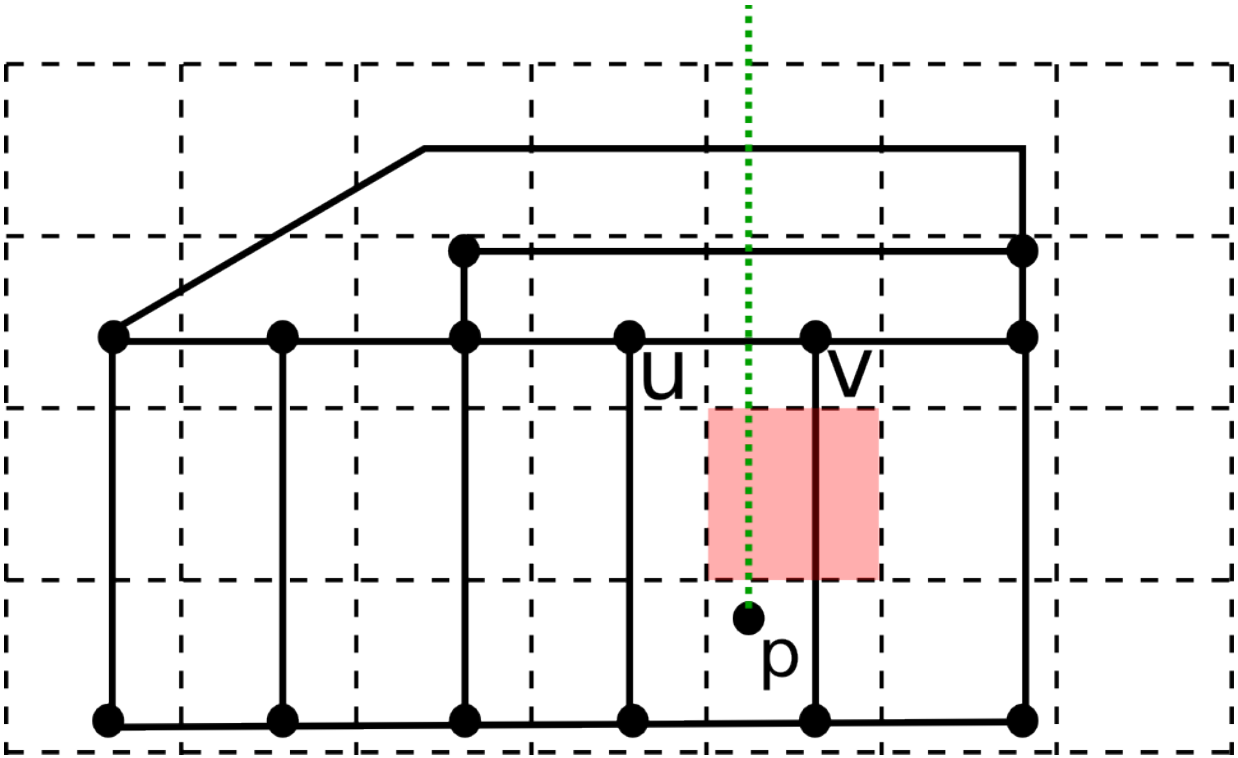
Locating vertices in other map

- Also implemented using a uniform grid.
- Given *p*, find the lowest edge above *p*.



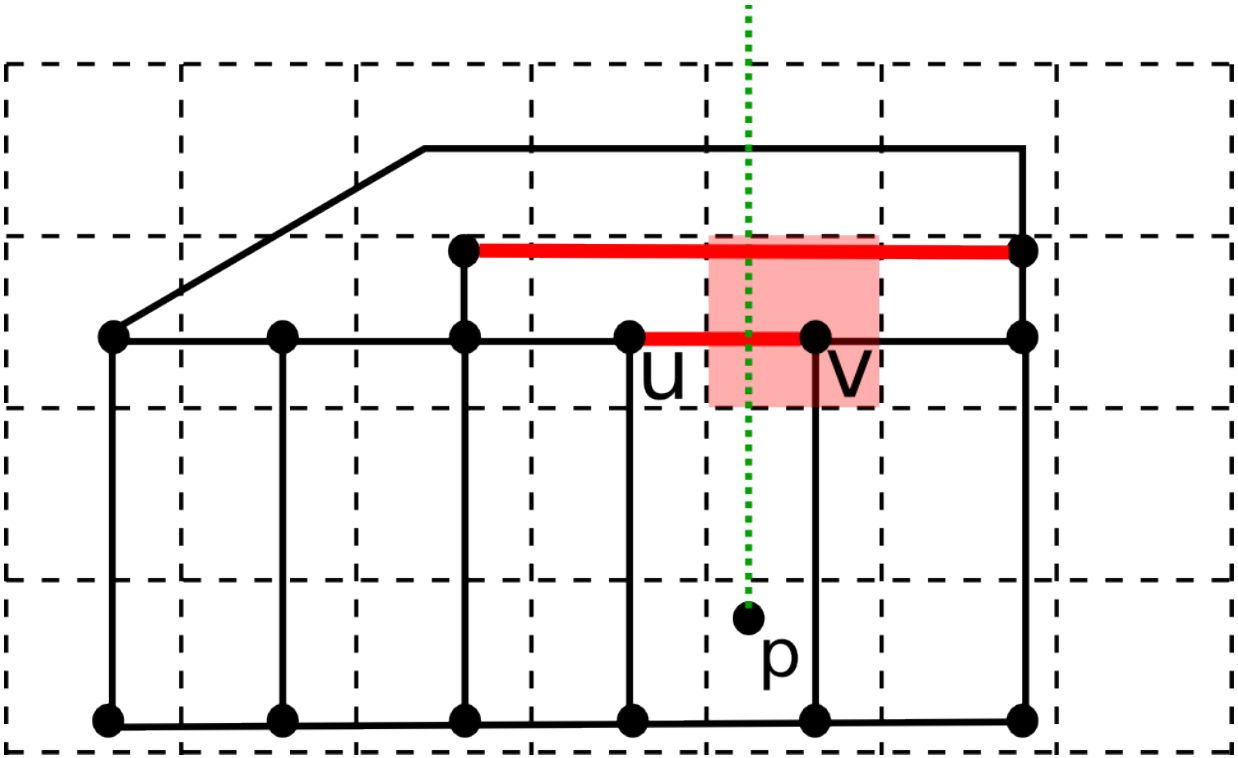
Locating vertices in other map

- Also implemented using a uniform grid.
- Given *p*, find the lowest edge above *p*.



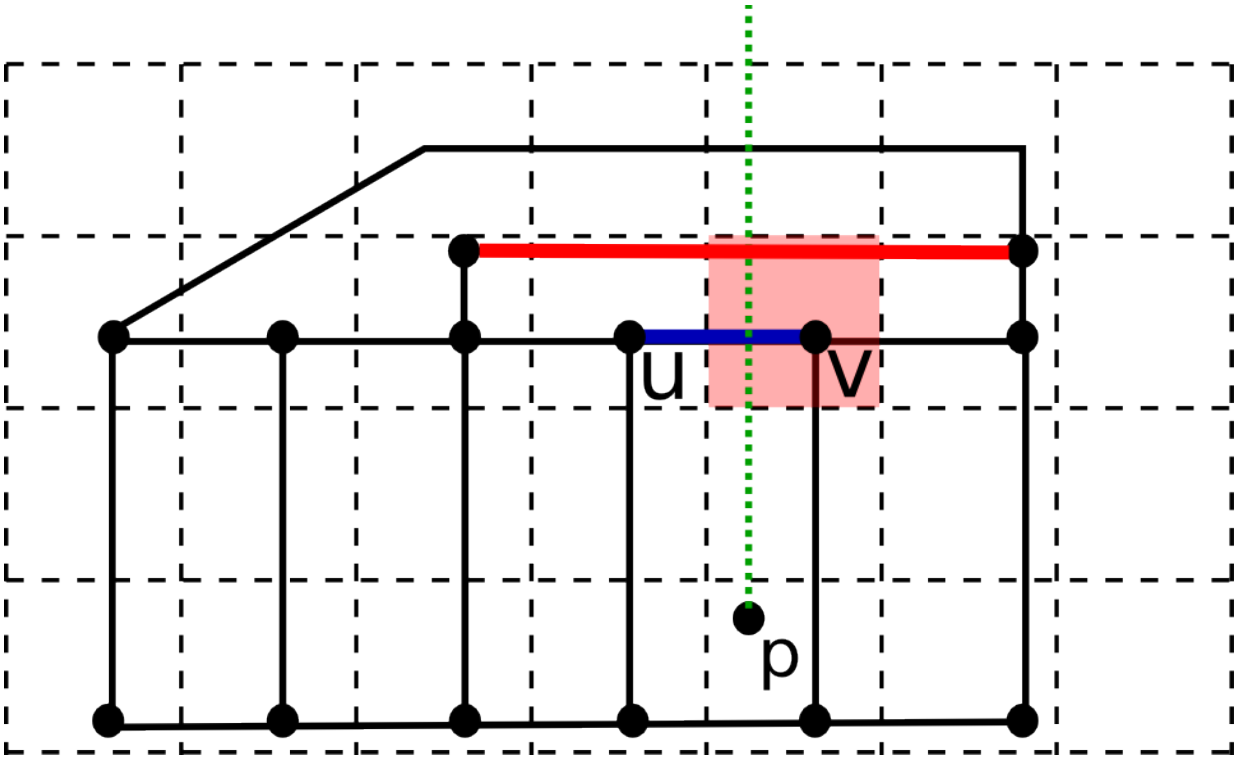
Locating vertices in other map

- Also implemented using a uniform grid.
- Given *p*, find the lowest edge above *p*.



Locating vertices in other map

- Also implemented using a uniform grid.
- Given *p*, find the lowest edge above *p*.



Computing output polygons

- Edges of the output polygons → computed based on input edges.
- For each input edge → three scenarios.

Computing output polygons

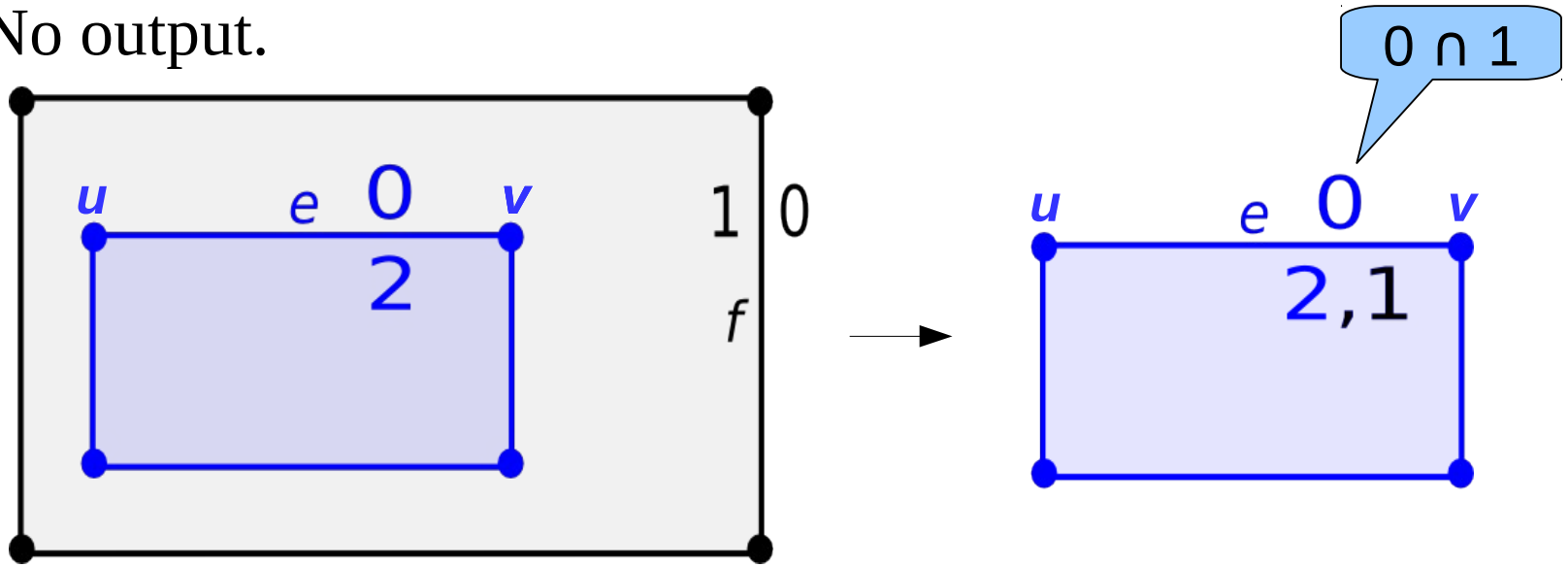
No intersection.

1 - edge completely inside a polygon (ex: *e*).

- Create output edge.

2 - edge completely outside a polygon (ex: *f*).

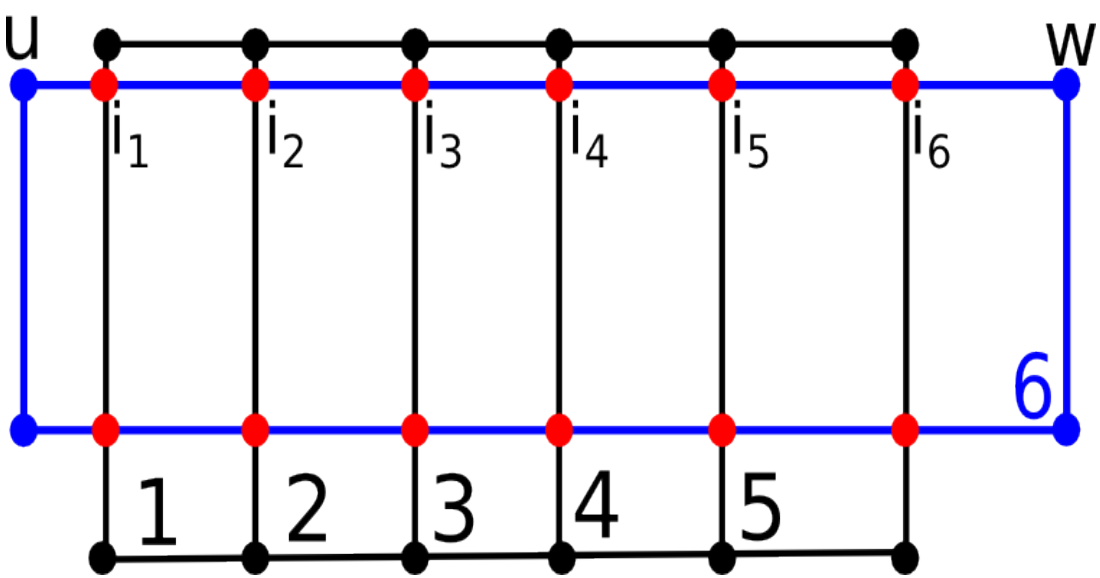
- No output.



Computing output polygons

3 – edge *e=(u,w)* with intersections.

- *e* is divided into segments.
- Output edge ↔ segment inside polygon.

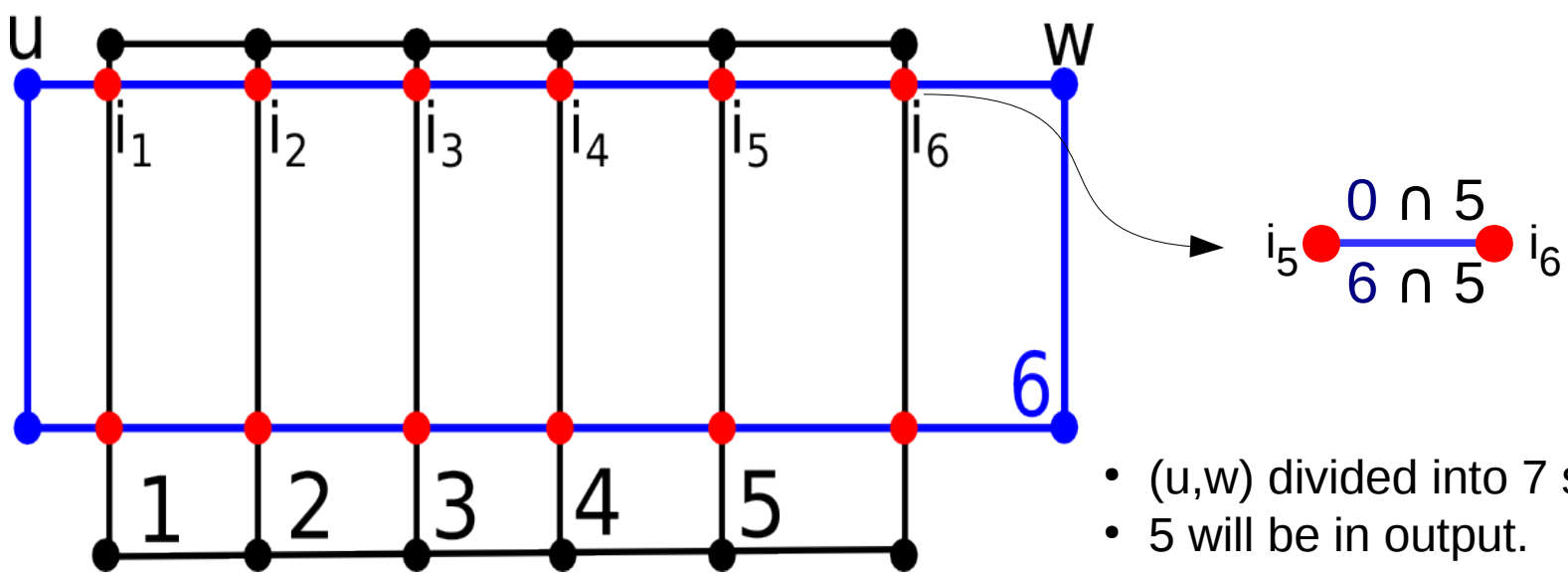


- (u,w) divided into 7 segments.
- 5 will be in output.

Computing output polygons

3 – edge *e=(u,w)* with intersections.

- *e* is divided into segments.
- Output edge ↔ segment inside polygon.

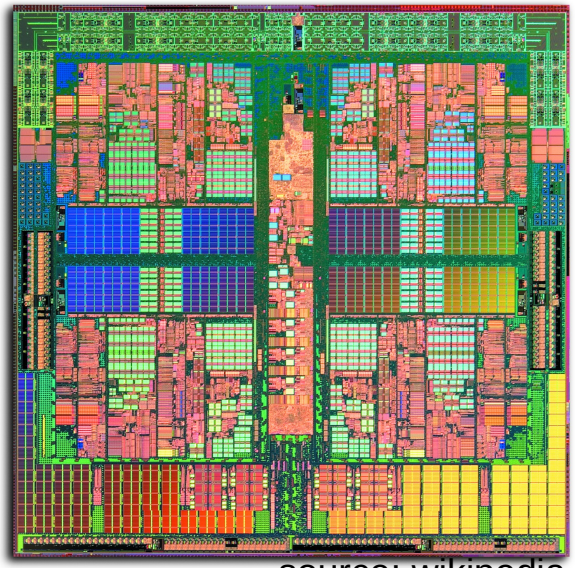


- (u,w) divided into 7 segments.
- 5 will be in output.

Parallel implementation

- 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.

- Most of computers: multicore → OpenMP.



source: wikipedia

Implementation details

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

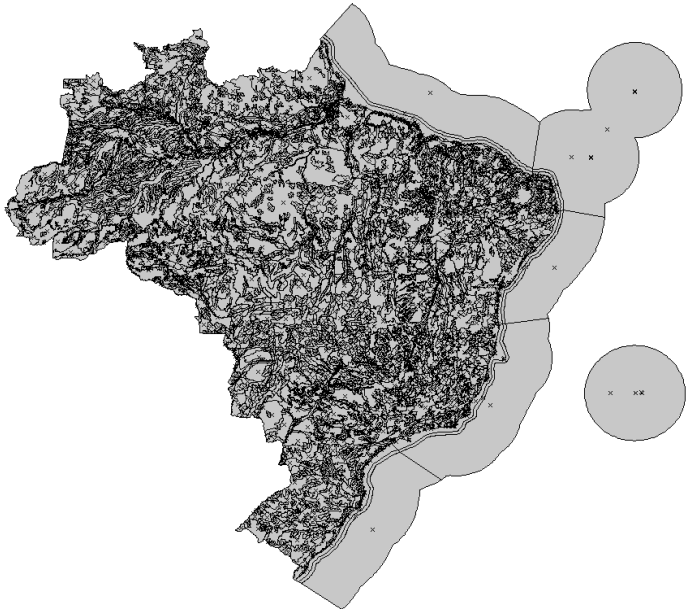
Experimental results

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

Experimental results

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

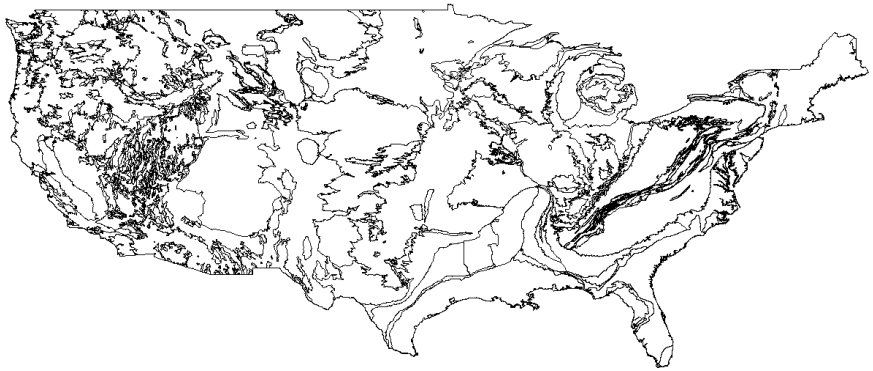
- BrCounty: 342,738 vertices, 2,959 polygons
- BrSoil: 258,961 vertices, 5,567 polygons.

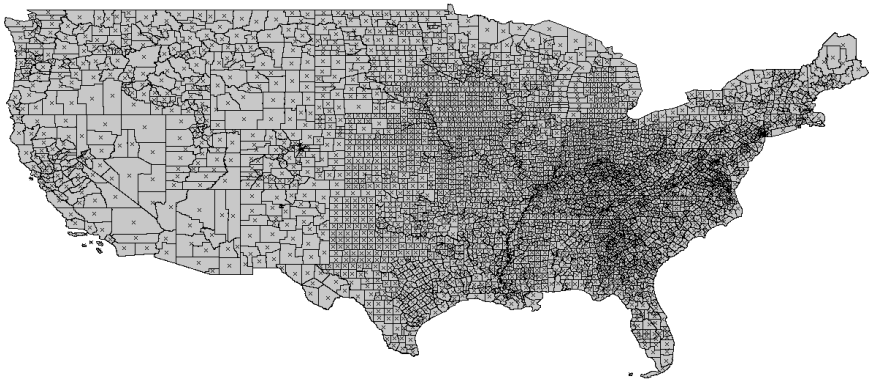


Experimental results

- 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





Experimental results

- Processing time vs uniform grid size.
- Maps overlaid with themselves to stress-test methods.
- Too big grids → # of intersection tests may increase.

Map 1	Map 2	# intersections	Uniform grid size 1000 Time(s)	1000 Tested inters.	2000 Time(s)	2000 Tested inters.
BrCounty	BrCounty	105,754	12.2	2,992,563	11.5	2,074,077
BrSoil	BrSoil	56,246	8.0	2,064,144	7.4	1,544,943
BrCounty	BrSoil	20,860	5.9	593,711	6.1	256,092
UsAquifers	UsAquifers	50,329	42.0	16,277,272	19.5	6,822,077
UsCounty	UsCounty	300,511	1047.9	802,175,027	500.8	392,661,330
UsCounty	UsAquifers	11,744	53.9	18,151,603	33.8	5,608,549

Map 1	Map 2	# intersections	Uniform grid size 8000 Time(s)	8000 Tested inters.	16000 Time(s)	16000 Tested inters.
BrCounty	BrCounty	105,754	14.9	2,242,946	23.8	3,289,793
BrSoil	BrSoil	56,246	11.2	1,939,365	20.2	2,988,865
BrCounty	BrSoil	20,860	9.5	93,739	16.2	74,192
UsAquifers	UsAquifers	50,329	11.9	2,256,347	17.3	2,163,809
UsCounty	UsCounty	300,511	171.8	103,311,854	124.4	56,720,186
UsCounty	UsAquifers	11,744	28.3	812,423	34.1	390,071

Experimental results

- 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

Experimental results

- 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

Conclusions and future work

- Rat-overlay is an efficient method.
- Use slow (but precise!) arithmetic. However, the performance is comparable with GRASS.
- Parallelizable algorithm → use computing power of modern computers.

- Future work
  - Compare performance against other methods.
  - Compare the quality of the output.
  - Analyze the uniform grid.

Thank you!

Any questions or suggestions?

Acknowledgement:

Contact:

W. Randolph Franklin: mail@wrfranklin.org
Salles V. G. de Magalhaes: vianas2@rpi.edu