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
  - $\text{Neighbor}(S)$: 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).
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!

![Diagram showing map generalization](image)
Previous work: map generalization

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

  ![Diagram](image)

  • Solution: dummy points.

  ![Diagram](image)

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

  ![Diagram](image)

• Also solved with dummy points.

  ![Diagram](image)
Previous work: map generalization

- For efficiency: uniform grid.
- Polylines points & control points $\rightarrow$ 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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># input points</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8531</td>
<td>3×10^4</td>
<td>3×10^4</td>
<td>3×10^5</td>
<td>4×10^6</td>
<td></td>
</tr>
<tr>
<td>Input reading</td>
<td>10</td>
<td>22</td>
<td>29</td>
<td>257</td>
<td>37092</td>
<td></td>
</tr>
<tr>
<td>Unif. grid init.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>24</td>
<td>1472</td>
<td></td>
</tr>
<tr>
<td>Simp. (Grid-Gen2)</td>
<td>2</td>
<td>15</td>
<td>13</td>
<td>435</td>
<td>23759</td>
<td></td>
</tr>
<tr>
<td>Simp. (Grid-Gen)</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>54</td>
<td>3481</td>
<td></td>
</tr>
<tr>
<td>Output writing</td>
<td>6</td>
<td>21</td>
<td>21</td>
<td>170</td>
<td>1817</td>
<td></td>
</tr>
</tbody>
</table>
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 $\rightarrow$ roundoff errors.
- Big amount of data $\rightarrow$ 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: \((\text{id}, \#\text{vertices}, \text{node}_0, \text{node}_1, \text{pol}_{\text{left}}, \text{pol}_{\text{right}})\)

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 $\rightarrow$ 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.

<table>
<thead>
<tr>
<th>Map 1</th>
<th>Map 2</th>
<th># intersections</th>
<th>Grid size</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Serial</td>
<td>Parallel</td>
</tr>
<tr>
<td>BrCounty</td>
<td>BrCounty</td>
<td>105,754</td>
<td>2,000</td>
<td>34.5</td>
</tr>
<tr>
<td>BrSoil</td>
<td>BrSoil</td>
<td>56,246</td>
<td>2,000</td>
<td>23.3</td>
</tr>
<tr>
<td>BrCounty</td>
<td>BrSoil</td>
<td>20,860</td>
<td>1,000</td>
<td>16.1</td>
</tr>
<tr>
<td>UsAquifers</td>
<td>UsAquifers</td>
<td>50,329</td>
<td>8,000</td>
<td>37.2</td>
</tr>
<tr>
<td>UsCounty</td>
<td>UsCounty</td>
<td>300,511</td>
<td>16,000</td>
<td>625.5</td>
</tr>
<tr>
<td>UsCounty</td>
<td>UsAquifers</td>
<td>11,744</td>
<td>8,000</td>
<td>67.5</td>
</tr>
</tbody>
</table>
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.

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