Exactly computing map overlays using rational numbers

Salles V. G. Maghalães and W Randolph Franklin

Rensselaer Polytechnic Institute, Troy NY

Oct 2014

We present an algorithm for overlaying two maps that is not hindered by roundoff errors and the resulting topological impossibilities. From two input maps containing polygons separated by polyline edges, the overlay is a map, each of whose polygons is the intersection of a polygon from each input map. A roundoff error caused by the finite precision of floating point numbers can cause a point to be computed to be on the wrong side of an edge. That can cause an edge intersection to be missed, or the point to be computed to be in the wrong polygon, leading to a topological inconsistency in the output map. This gets worse when the inputs are bigger, or have slivers. We completely avoid this problem by representing coordinates as rational numbers.

Now, intersections have no roundoff errors. Rational numbers are well known to computer scientists, and packages exist, but there are challenges, such as their speed and the fact that the packages might not have been used on such large datasets. Our solution, that uses a uniform grid to accelerate the computation, is very efficient and, as tests showed, it can overlay maps faster than the widely used GRASS GIS.

Keywords: map overlay; computational geometry; algorithms

Introduction

With the recent advances in GIS technologies, a huge amount of high quality geographical data has become available. Thus, it is necessary to develop efficient and precise algorithms to perform operations in these data. One of the most important kinds of operation is the overlay of maps, where two vectorial maps are superimposed and a resulting map containing the intersections of the polygons from the input data is created. This operation has several applications like, for example, overlaying a map containing counties boundaries with a map containing watershed boundaries, which would result in a map containing the regions of each county that drain into the different watersheds.

A particular challenge in the overlay problem are the topological impossibilities that are often created due to roundoff errors caused by the finite precision of floating point numbers stored in computers. These problems are even worse when big amounts of data are processed and when input edges almost coincide. It is important to notice that edges coincidence is not rare in real maps. For example, borders of counties are frequently defined based on natural features such as rivers. Thus, edges from a county map may coincide with edges in a soil map, where rivers form borders for different kinds of soil.

This paper presents *Rat-overlay*, an algorithm that uses rational numbers to perform the exact map overlay, completely avoiding roundoff errors. Using rational number ensures that intersections are computed exactly and avoids the formation of slivers in the map. However, there is a challenge for using these kinds of numbers, since operations with rationals present a big overhead if compared against hardware-implemented floating point operations. Also, the result of an addition or multiplication has about the number of digits of the two operands combined and, thus, consecutively performing mathematical operations may require a big amount of space and computation. Therefore, operations such as accumulating the area of a polygon are not efficient with rational numbers.

A uniform grid is used to efficiently compute the edge intersections, reducing the amount of operations performed with the rational numbers. The edges and vertices resulting from the intersections are, then, classified and, finally, the overlay areas and output polygons are computed. Also, to better use the computing power of current multiprocessors, *Rat-overlay* was parallelized using OpenMP API.

As presented in the tests, *Rat-overlay* was able to efficiently compute the exact overlay of maps. In fact, the use of parallel programming allowed *Rat-overlay* to process maps faster than the approximate overlay algorithm available in GRASS GIS (GRASS Development Team 2014).

The Rat-overlay method

Map data format

In this work, each input map is represented as a planar graph where the edges represent the regions boundaries and each region is labeled with an identification number. By convention, the exterior region is represented by the identifier *0*.

For simplicity, each sequence of adjacent edges that bounds two given regions is grouped in a chain and each chain is labeled with the following information: $(id,\#edges,node_0,node_1,pol_{left},pol_{right})$, where *id* represents the unique identification number of the chain, *#edges* represents the number of edges in the chain, $node_0$ and $node_1$ are identification numbers for each of the endpoints and pol_{left} and pol_{right} represents, respectively, the identification numbers of the polygons in the left and in the right side of the chain. Each chain label is followed by *#edges*+1 coordinates of the vertices.

Observe that only the chains are explicitly stored in the map file. Also, it is important to mention that a region may be composed of several polygons and, in this situation, the polygons representing the same region have the same identification numbers. Figure 1 presents an example of a map composed of 2 regions (one of them is composed of two polygons) and the corresponding chains that are used to represent this map.

Overlaying two maps

The first step of the algorithm consists in finding all the intersections between the edges of the first input map with the edges of the other input map. As it will be explained in the next section, this computation is accelerated using a uniform grid to reduce the amount of pair of edges tested for intersection.

Then, for each vertex of the input maps, the algorithm computes in which polygon of the other map this vertex is located. This information will be used in the computation of the output polygons and of their areas. As it will be explained later, this step of the algorithm is also accelerated using a uniform grid.

After locating the vertices, *Rat-overlay* computes the output polygons' areas. It is important to mention that this step is not necessary for computing the resulting polygons and, as explained in Section *Using rational numbers*, the areas are computed mainly for validating the correctness of the input and of *Rat-overlay*. The areas are obtained using a strategy proposed by Franklin et. al (1994), that considers that each polygon edge is divided in two directed ‘half-edges’ (each one ends in one endpoint of the original edge) and uses these ‘half-edges’ to compute the areas of the polygons.

The basic idea is ‘that some properties, such as area and perimeter, can be calculated independently for each half-edge and, then, summed’ (Franklin et. al 1994). Notice that in this step of *Rat-overlay*, the output polygons were not computed yet: similarly to Franklin et al.'s algorithm, the area of the output polygons are computed without explicitly computing the polygons. This is performed by dividing the original edges in the intersections of the two maps and, then, accumulating the area of each half-edge (derived from original edges or from divided edges) in the corresponding output polygon. Since the areas do not need to be computed exactly (and, as mentioned later, this is usually infeasible), they are computed and accumulated using primitive floating point numbers.

To compute the output polygons, each edge from the input maps is analyzed and the outputs generated from these edges are computed. An input edge *e* may be classified in three different categories: *e* is completely inside a polygon of the other map, *e* is not inside any polygon of the other map, *e* intersects edges from the other map.

If an edge *e* does not intersect any other edge in the other input map, *e* may be completely inside one polygon in the other map or completely outside the polygons of the other map. In the first situation, *e* will be an edge of the output map and, in the other situation, *e* will not be in the output. Figure 2 illustrates these two situations: edge *e* is inside polygon *1* of the other map (represented by black edges) and, thus, in the resulting map it will be in the boundary between the exterior polygon and the polygon relative to the intersection of polygon *1* of one map with polygon *2* of the other one. Edge *f*, on the other hand, is not inside any polygon of the map represented in black and, thus, it will not be in the output map.

To determine if *e* is inside a polygon in the other map, our algorithm checks in which polygon of the other map one of the endpoints *v* of *e* is. If v is in polygon *0* (the outside of the map), *e* is completely outside the other map. Otherwise *e* will be completely inside the other map.

If $e=(u,w)$ intersects $k \geq 1$ edges of the other map in points $i_1, i_2, ..., i_k$, this will divide *e* in $k+1$ edges $e_1=(u, i_1)$, $e_2=(i_1, i_2)$, $e_3=(i_2, i_3)$, ... $e_k(i_{k-1}, i_k)$, $e_{k+1}=(i_k, w)$. To determine which of these edges will be in the output map, the midpoint m_j of each edge e_j is analyzed: if m_j is outside a polygon of the other map, the polygons in both sides of the corresponding edge e_j will not intersect any polygon in the other map and, thus, e_j will not be in the output. Otherwise, if m_j is inside a polygon *p* of the other map, e_j will be in the output map since the polygons in both sides of e_j will intersect with *p*.

Figure 3 presents an example of situation where an edge $e=(u,w)$ from one map (represented in blue) intersects edges from another map (black) in several points (detached in red). The edge $e_2=(i_2,i_3)$ resulting from the intersections has midpoint m_2, that is inside polygon *2* of the other map. Since the polygon in the right and left sides of *e* are, respectively, *6* and *0*, the output polygon in the right side of e_2 will correspond to the intersection of polygon *6* with *2* and the polygon in the left side of e_2 will be the 'outside' polygon (created from the intersection of polygons *2* and *0*). Edge $e_6=(i_6,w)$, on the other hand, will not be in the output map since its midpoint m_6 is not inside any polygon of the other map and, therefore, both of its sides will be outside of polygons in the resulting map.

Implementation details

As mentioned in the previous section, a uniform grid is used to accelerate some steps of the algorithm. The basic idea, proposed by Franklin et al. (1989), is to create a *N x M* grid and superimpose this grid with the two input maps such that all input edges are inside the grid. Then, the intersections can be computed in each grid cell. The computation is accelerated because only pairs of edges that lie inside one cell (or intersect it) need to be tested for intersection. In our implementation, the dimensions of the grid are defined by the user.

Figure 4 presents an example where the intersections between map *1* and map *2* are computed using a uniform grid with 4 *x* 7 cells. Notice that, during the computation of the intersections that lie in the detached cell (in light red), only 3 edges from map *1* need to be tested for intersection with the only edge of map *2* that is in this cell.

The algorithm to determine in which polygon of a given map a point *p* is located is also implemented using a uniform grid. The basic idea is to find the lowest edge *e* that is above *p* and intersects a vertical line *l* that crosses *p*. The polygon where *p* is located will be the polygon that is in same the side of *e* as *p*.

Supposing *p* is in cell *C* of the uniform grid, the algorithm initially tries to find the closest edge higher than *p* that crosses *l*. If there is no such an edge in *C*, this process is repeated in the next cells higher than *C*. If neither *C* nor the other cells higher than *C* contains an edge that crosses *l*, then *p* is in the 'outside' polygon (that is labeled *0*). In the example in Figure 5, no edge in the cell *C* where *p* is located and in the cell immediately above *C* crosses *l* and is above *p*. However, there are two edges in the cell detached in red that crosses *l*. Since the edge *e=(u,v)* is the lowest of these edges, the algorithm determines that *p* is in the polygon in the right side of *e=(u,v)*.

Simulation of simplicity (Edelsbrunner and Mücke 1990) is used to avoid degenerate cases in the computation of the intersections and, also, to break ties in the location of the vertices. The idea is that, during the computations, the algorithm 'pretend' that the first input map is slightly below and to the left of the second input map. By doing that, no edge from one map will coincide with edges from the other map during the intersection computation.

Parallel implementation

The performance of *Rat-overlay* in parallel computers with shared-memory architecture was improved by using OpenMP API. First, during the uniform grids creation, the edges from both methods are added in parallel to the corresponding grids. Despite there is no data dependency in this processing, it is necessary to use synchronization methods to ensure data consistency when the edges are inserted in the grid data structure.

Also, all vertices of each map need to be located in the polygons of the other map to compute the polygon areas and the output polygons. Since there is no data dependency in the processing of each vertex, they are located in the other map in parallel.

Furthermore, the intersections are found in parallel: notice that the intersections are detected independently in each cell from the uniform grid. Thus, these cells can be processed in parallel.

Finally, the output polygons are also computed in parallel. As explained previously, the input edges are classified based on their intersections with edges in the other map and the output edges are created based on this classification. Therefore, each input edge can be processed independently in parallel.

Using rational numbers

Computing in the algebraic field of the rational numbers over the integers, where the integers are allowed to grow as long as necessary, allows the traditional arithmetic operations, addition, subtraction, multiplication, and division, to be computed exactly, with no roundoff error. The cost is that the number of digits in the result of an operation is about equal to the sum of the numbers of digits in the two inputs.

E.g., $\frac{214}{433}+\frac{659}{781}=\frac{214524814}{338173}$. The inputs have 12 digits in all; the output has 12 digits.

Casting out common factors helps, but that is rare. E.g., we can cast out a common factor of two when the numerator and denominator are both even, which occurs 1/4 of the time.

Ever longer numbers is not a problem when computing the sign of the determinant of a 3*x*3 matrix. That is the major part of determining whether or not two lines intersect. That is the operation whose accuracy is critical in overlaying two maps to compute the output polygons. This is why it is feasible to use rational numbers to overlay maps.

In contrast, computing the area of a polygon is an example of an operation that would not be amenable to computing over the rationals. The area is a function of all the polygon's vertices. Therefore, the number of digits in the area's numerator and denominator would be close to the sum of the numbers of digits in all the polygon's vertices. Luckily, we do not need to know the polygons' exact areas. Indeed, the only application of areas in our program is as a sanity check. The computed area is the sum of many large numbers that are almost absolutely equal but have opposite signs. If any term is omitted, the erroneously computed area will be very large, and with probability one-half, negative. The absence of such obvious errors increases confidence that our program is correct.

Rational numbers are also not sufficient to exactly compute intersections of circles and straight lines, because the result is usually an irrational number. If that were a concern, we could compute them exactly by using the field of rational numbers extended by the square root operation. Here, numbers such as $\frac{1+\sqrt{2}}{\sqrt{3}-5}$ can be represented exactly, and not as floating approximations. Tools such as the Computational Geometry Algorithms Library (CGAL) and Mathematica can do this; but they are big and slow.

Experimental results

To evaluate *Rat-overlay*, it was implemented in C++ using GMPXX (Granlund et al. 2014) as the multiple precision arithmetic package, compiled with g++ 4.8.2 and tested in a machine with the following configuration: dual Intel Xeon® E5-2687 processor (totaling 16 cores, that are able to run 32 threads using Intel Hyper-threading technology), 128 GiB of RAM memory and Linux Mint 17 operating system. Unless otherwise stated, all tests with *Rat-overlay* were performed using its parallel version configured to execute using *32* threads.

Tests were performed using two datasets from Brazil, distributed by IBGE (the Brazilian geography agency) and two datasets from the United States, obtained from the United States Census and National Atlas webpages. All these maps are distributed in shapefile format and, thus, they needed to be converted to the format (that will be described later) used as input to *Rat-overlay*.

- BrSoil: Map representing different kinds of soils in Brazil. It contains 258,961 vertices, 5567 polygons and was obtained in IBGE's website (IBGE 2014a).
- BrCounty: Represents the Brazilian county subdivision. This map contains 342,738 vertices, 2959 polygons and was also downloaded from IBGE's website (IBGE 2014b).

- UsAquifers: Contains polygons representing the aquifers that supply water to the United States. This dataset contains 195,276 vertices, 3552 polygons and was obtained in the National Atlas' website (National Atlas 2014).
- UsCounty: Represents the county subdivision in the United States. The shapefile contains 3,648,726 vertices, 3110 polygons and was downloaded from the United States Census' website (U.S. Census Bureau 2014). Since, differently from UsAquifers, this map contained several islands (such as Guam) that were far from the mainland, the insular part of the map was removed.

In order to evaluate how the computation of the exact overlay impacts the performance of the proposed method, it was compared against GRASS GIS 6.4 (GRASS Development Team 2014) *v.overlay* module. Tests were performed overlaying the pairs of maps of the same countries. Also, we overlaid each map with itself in order to stress-test the algorithms in situations where all edges coincide.

In the first set of tests, we evaluated the influence of the uniform grid size in the running time of the algorithm. Table 1 presents the total running-time (including I/O) for different grid sizes (all grids used in these tests are square). Also, the number of edge-edge intersections in each map and the total number of edges pairs tested for intersection in each test case is included.

Notice that, for the Brazilian datasets, the best results were always obtained using small grids (1000^2 and 2000^2 cells), while in the datasets from the United States the best results were obtained with larger ones. This may be explained because, as it can be seen in Table 1, increasing the grid size does not necessarily reduces the amount of intersections that need to be tested (for example, when the BrCounty map is overlaid with itself, the number of tested intersections increase if the grid size increases from 2000^2 to higher values). This happens because, using too large grids, the size of each grid cell is smaller and, thus, one edge may intersect several cells, which may increase the number of intersections tested. Furthermore, even if the number of intersections that need to be tested decrease, this reduction may not balance the overhead of using a larger grid (this could explain the reason that made the 8000^2 grid be better for the overlay of maps UsCounty with UsAquifers than the $16,000^2$ grid).

Table 2 presents a comparison between a sequential version of *Rat-overlay*, a parallel one and the *v.overlay* module of GRASS GIS (that is also a sequential implementation). The values for the uniform grid size were chosen based on the configurations that performed better in the tests presented in Table 1 and the *v.overlay* parameters were configured using the default values suggested by GRASS GIS.

As it can be noticed, the parallel version of *Rat-overlay* was faster than GRASS in all test cases. In fact, overlaying BrCounty with BrSoil was almost 14 times faster using our method than using GRASS. The sequential version, on the other hand, was slower than GRASS in the two test cases with the largest amount of edges intersections. This indicates that, even though we compute the exact overlay using multiple precision arithmetic (that is much slower than hardware-implemented floating point operations), our method can present a competitive performance if compared against a method from a widely used GIS.

It is important to observe that, despite we executed the parallel implementation using *32* threads, its speedup ranged from 3 to 5 times if compared against the sequential implementation. This may be explained because we needed to use several critical sections to ensure that concurrent accesses to the data structures were performed safely. Also, despite the method was executed using 32 threads, the computer where the tests were performed has only 16 physical cores. Furthermore, as it will be shown in the next tests, I/O corresponds to a significant amount of the processing time. Thus, according to Amdahl's law, I/O may limit the speedup of the parallel implementation, since disk access performance usually cannot be improved using parallel programming.

Finally, Table 3 presents the amount of time that *Rat-overlay* spent in each step of the overlay computation. It is interesting to observe that the bottleneck of the algorithm varies for different test cases. For example, the time spent performing I/O ranges from 16% to 38% of the total time, being the algorithm's bottleneck in two of the test scenarios. Edges intersections computation, on the other hand, represents a small percentage of the execution time in some scenarios (for example, 7% of the total time during the overlay of UsCounty with UsAquifers), while in the overlay of UsCounty with itself it corresponds to almost half of the execution time, which can be explained because when the maps are the same, the amount of edges of each map in each grid cell is the same, which maximizes the number of pair of edges to be tested for intersection.

Conclusion

This work proposed *Rat-overlay*, an efficient method that uses rational numbers to compute the exact overlay between two maps. As tests showed, even though *Rat-overlay* performs computation using multiple precision arithmetic (that is much slower than hardware-implemented operations with floating point numbers), its performance is competitive if compared against the approximate overlay method present in the widely used GRASS GIS.

Furthermore, *Rat-overlay* can be implemented using parallel programming techniques. As shown in the tests, an OpenMP implementation was able to achieve from 3 to 5 times of speedup if compared against the sequential implementation.

As future work, we intend to compare the performance of *Rat-overlay* against other algorithms (such as the algorithms implemented in ArcGIS®). Also, we intend to compare the quality of the exact output obtained by *Rat-overlay* against approximate overlays obtained by other methods. Finally, another future work is to analyze in more details which factors influence the optimal size of the uniform grid that should be used for different input maps. By doing that, the objective is to develop some heuristic to automatically determine the adequate grid size for each execution.

Acknowledgment

Removed because of the blind review.

References

Edelsbrunner, Herbert and Mücke, Ernst Peter. 1990. “Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms.” In ACM Transactions on Graphics (TOG), vol. 9, no. 1, 66–104, ACM.

Franklin , Wm Randolph, Sivaswami , Venkateshkumar, Sun, David, Kankanhalli , Mohan, and Narayanaswami , Chandrasekhar. 1994. “Calculating the area of overlaid polygons without constructing the overlay.” In Cartography and Geographic Information Systems, vol. 21, no. 2, 81–89, Taylor & Francis.

Franklin , Wm Randolph, Sun, David, Zhou, Meng-Chu, and Wu, Peter YF. 1989. “Uniform grids: A technique for intersection detection on serial and parallel machines.” In Proceedings of Auto Carto 9: Ninth International Symposium on Computer-Assisted Cartography, 100–109.

Granlund, Torbjörn and the GMP development team. 2014. “GNU MP: The GNU Multiple Precision Arithmetic Library.” Accessed August 2014. http://gmplib.org/.

GRASS Development Team. 2014. “Geographic Resources Analysis Support System (GRASS GIS) Software.” Open Source Geospatial Foundation. Accessed August 2014. http://grass.osgeo.org

IBGE. 2014. “Mapa de solos do Brasil.” Accessed August 2014. ftp://geoftp.ibge.gov.br/mapas tematicos/mapas murais/shapes/solos

IBGE. 2014. “Malha municipal digital 2007.” Accessed August 2014. ftp://geoftp.ibge.gov.br/malhas digitais/municipio

National Atlas. 2014. “Principal aquifers of the 48 conterminous United States, Hawaii, Puerto Rico, and the U.S. Virgin Islands.” Accessed August 2014. http://nationalatlas.gov/mld/aquifrp.html

U.S. Census Bureau. 2014. “United States Census Shapefiles.” Accessed August 2014. https://www.census.gov/cgi-bin/geo/shapefiles2013/main

			Uniform grid size							
Map 1	Map 2	Inters.	1,000		2,000		8,000		16,000	
			Time	Test. int.	Time	Test. int.	Time	Test. int.	Time	Test. int.
BrCo.	BrCo.	1x10^{5}	12	3x10^{6}	11	2x10^{6}	15	2x10^{6}	24	3x10^{6}
BrSoil	BrSoil	6x10^{4}	8	2x10^{6}	7	2x10^{6}	11	2x10^{6}	20	3x10^{6}
BrCo.	BrSoil	2x10^{4}	6	6x10^{5}	6	3x10^{5}	9	9x10^{4}	16	7x10^{4}
UsAq.	UsAq.	5x10^{4}	42	2x10^{7}	19	7x10^{6}	12	2x10^{6}	17	2x10^{6}
UsCo.	UsCo.	3x10^{5}	1,048	8x10^{8}	501	4x10^{8}	172	1x10^{8}	124	6x10^{7}
UsCo.	UsAq.	1x10^{4}	54	2x10^{7}	34	6x10^{6}	28	8x10^{5}	34	4x10^{5}

Table 1. Comparison of the processing time (in seconds) to overlay the map in column *Map 1* with the map in column *Map 2* considered different uniform grid sizes. Column *Test. int* presents the number of pairs of edges tested for intersection and column *Inters.* presents the number of intersections effectively found.

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

Table 2. Comparison of the processing time (in seconds) of the parallel and sequential versions of *Rat-overlay* against GRASS *v.overlay* module.

Map 1	BrCounty	BrSoil	BrCounty	UsAquifers	UsCounty	UsCounty
Map 2	BrCounty	BrSoil	BrSoil	UsAquifers	UsAquifers	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

Table 3. Comparison of the amount of time (in seconds) spent by *Rat-overlay* to perform the main steps of the overlay computation.

Figure 1. Example of a map and the chains stored in the corresponding map file. Observe that each chain is displayed using different colors in the map.

Figure 2. Example of two maps (one represented in black and another one represented in blue) whose edges do not intersect.

Figure 3. Example of two maps (one represented in black and another one represented in blue) that contains edge intersections. Points represented in red are the intersections of the edges from the two maps and points in green represents the midpoints of some edges created by the intersections.

Figure 4. Example of a 4 *x* 7 uniform grid used to accelerate the computation of the intersections between the edges from map 1 (in black) with the edges from map 2 (in blue).

Figure 5. Using a uniform grid to determine in which polygon a point *p* is.