Filtering with Raster Signatures

Leonardo Guerreiro Azevedo[1], Ralf Hartmut Güting[2], Rafael Brand Rodrigues[1], Geraldo Zimbrão[1,3], Jano Moreira de Souza[1,3]

[1]Computer Science Department, Graduate School of Engineering, Federal University of Rio de Janeiro, PO Box 68511, ZIP code: 21945-970, Rio de Janeiro, Brazil, +55-21-2562-8694

[2]LG Datenbanksysteme für neue Anwendungen, FB Informatik,

Fernuniversität Hagen, D-58084 Hagen,Germany, +49-2331-987-4279

[3]Computer Science Department, Institute of Mathematics,
Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, +55-21-2562-8694

Email: {azevedo, brand, zimbrao, jano}@cos.ufrj.br, rhg@fernuni-hagen.de

ABSTRACT

Efficient evaluation of spatial queries is an important issue in spatial database. Among spatial operations, spatial join is very useful, intersection being the most common predicate. However, the exact intersection test of two spatial objects is the most time-consuming and I/O-consuming step in processing spatial joins. On the other hand, the use of approximations can reduce the need for examining the exact geometry of spatial objects in order to find the intersecting ones. This work proposes a new raster approximation (Three-Color Raster Signature - 3CRS) for representing different data types (polygons, polylines and points), and to be used as filter in the second step of the Multi-Step Query Processor. We have also executed experimental tests over real datasets, the results having demonstrated the effectiveness of our approach.

Categories and Subject Descriptors

E.2 [**Data**]: Data Storage Representations – *Object representation*; H.2.8 [**Database Management**]: Spatial databases and GIS

General Terms

Algorithms, Management, Performance, Experimentation

Keywords

Spatial databases, GIS, Raster Approximation, Spatial Join, Multi-Step Query Processor, Three-Colors Raster Signature, Four-Colors Raster Signature

1. INTRODUCTION

The increase in storage capacity and the decrease of hardware prices have enabled applications to deal with large amounts of data, involving Gigabytes, Terabytes and even Petabytes of information. This characteristic is common to Spatial Databases where data usually have high complexity and are available in huge amounts.

Spatial data consists of spatial objects made up of points, lines, regions, rectangles, surfaces, volumes, and even data of higher dimension which includes time [13]. Examples of spatial data include cities, rivers, roads, counties, states, crop coverage, mountain ranges etc. It is often desirable to attach spatial with non-spatial attribute information. Examples of non-spatial data are road names, addresses, telephone numbers, city names, etc. Since spatial and non-spatial data are so intimately connected, it is not surprising that many of the issues that need to be addressed are in fact database issues.

There are numerous applications in the area of spatial database systems, such as: traffic supervision, flight control, weather forecasting, urban planning, route optimization, cartography, agriculture, natural resources administration, coastal monitoring, fire and epidemics control, precision agriculture and intelligent highways ([1], [8] and [14]). Each type of application deals with different features, scales and spatiotemporal properties.

Efficient evaluation of spatial queries is an important issue in spatial database. Among spatial operations, spatial join is one of the most useful. Intersection is the most common join predicate. Many works point the exact geometry test as the most time consuming operation regarding both I/O and CPU time. [6] show experimental results confirming that the exact geometry test, usually plane-sweep ([3] and [7]) is responsible for most of the CPU cost. The I/O cost associated with the exact geometry test is due to the access to the real representation of spatial objects, which can be very large. Spatial joins have been well studied in the literature, and there are many approaches to processing spatial join operations. Considering points, polylines and polygons as the three data types most commonly found in spatial databases, there are nine classes of different spatial joins. For its usefulness and complexity, the polygon join has been the most investigated, while the point join has been the less investigated because of its similarity with the relational join ([13]), while there are some proposals for processing polylines × polylines joins and polygon × polyline joins. In order to improve the efficiency, organization and study of indices and filters for spatial data, [6] proposed a three-step architecture for spatial join processing, named Multi-Step Query Processor. The main target of this architecture is to accelerate the most costly step by reducing the number of spatial objects left to be compared. Such a reduction is done by applying filters in previous steps.

This work proposes a new raster approximation suitable to performing spatial joins as a filter in the second step of the Multi-Step Query Processor, involving these three common data types (polygon, polyline and point) and the classes of different spatial joins involving them. We propose a raster signature named as Three Color Raster Signature (3CRS), based on the Four-Color Raster Signature (4CRS) proposed by [15]. The 3CRS signature has the main advantage of faster generation time and that can be used to represent polygons, polylines and points (without any specific characteristic). Besides, the same algorithm can be used to evaluate the join predicate involving these three data types. Also, the fast generation time allows on-the-fly signature generation. For instance, instead of storing the signature, it can be generated only when it is needed, saving storage space. In order to evaluate the effectiveness of our proposal, we executed spatial joins using 3CRS against the processing without using signatures and the processing using 4CRS. The experimental tests were executed over real data and the results demonstrated the effectiveness of the approach.

This paper is divided in sections, as follows: Section One is this introduction; Section Two surveys the related literature; in Section Three, we present our raster approximation and its implementation as the second step of the MSQP architecture; Section Four shows the experimental results; finally, in Section Five we present our conclusions.

2. ARCHITECTURES FOR PROCESSING SPATIAL JOINS

There are many approaches to processing spatial join operations. [16] emphasize that traditional approaches execute spatial join processing in two steps ([10] and [12]). They propose efficient algorithms to be used in the second step. In the two-step approach, presented in Figure 1.a, the first step employs a Spatial Access Method (SAM) in order to reduce the search space. The Minimum Bounding Rectangle (MBR) is normally used by SAM methods. This step does not have the result of the join operation as output. Instead, it provides a set of candidate pairs that correspond to a super-set of the solution, and that is sent to the second step. The second step is a refinement step where the pairs resulting from the first step are read from disk and have their geometries processed. This is the most costly step, requiring I/O time to seek and read the spatial objects from disk, and CPU time to compute the exact answer.

[6] propose a three-step architecture for spatial join processing named as Multi-Step Query Processor (MSQP), presented in Figure 1.b. In this architecture, another step is included between the first (SAM) and the second (Exact geometry processor) steps. The proposed step consists in comparing the candidate pairs from the first step using a geometric filter. The geometric filter uses a compact and approximated representation of the object, trying to retain its main characteristics. Examples of such proposals of object representations are: 4CRS ([15]), Convex Hull, 5C, RMBR and others found in [5]. As the result of this step we have three possibilities: pairs that belong to the solution (hit); pairs that do not belong to the solution (false hit); and, pairs where it is not possible to have a conclusive answer (inconclusive comparisons or candidate pairs). The latter are sent to the third step, the refinement step, where the pairs of objects are read from disk and have their geometries processed.

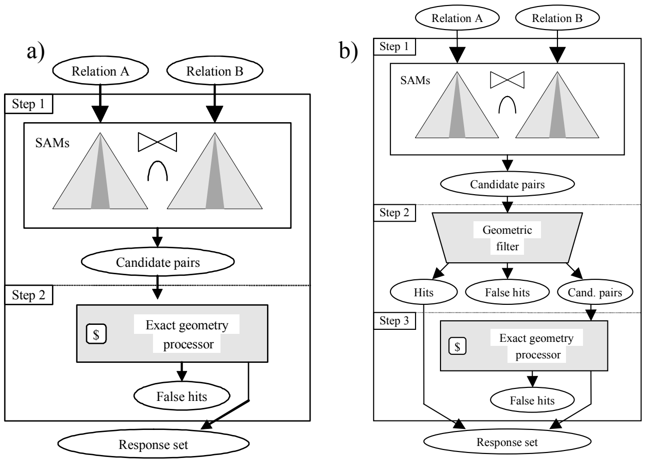


Figure 1. a) Two-step architecture for spatial join processing; b) Three-step architecture [6]

There are two main advantages to introducing the filter step. First, the approximation size is only a fraction of the spatial object size; therefore it can be stored in the index, together with the object MBR. Secondly, testing the intersection of two approximations requires less CPU time than testing two objects. The pair of objects that have a conclusive test (hit or false hit) are not sent to the third step.

3. THREE-COLOR RASTER SIGNATURE

In this section we will present the characteristics of 3CRS. Section 3.1 presents 3CRS main characteristics. Section 3.2 proposes an algorithm to generate 3CRS. A simple algorithm for changing the resolution of 3CRS is proposed in Section 3.3, and the algorithm to evaluate if two objects overlap using their 3CRS is presented in Section 3.4.

3.1 3CRS Characteristics

The 3CRS is based on 4CRS ([15]). 3CRS is a compact and approximated raster representation of objects upon a grid of cells that uses few colors. Each color represents an intersection type between the object and the cell (Table 1). Figure 2 presents an example of a 3CRS representation of a polygon. Actually, 3CRS is a 4CRS where the *Weak* and *Strong* cell types are replaced by an *Inconclusive* type. The *Weak* 4CRS cell type represents that the polygon has an intersection equal or less than 50% with the cell, and the *Strong* type represents an intersection greater than 50% and less than 100%. The 3CRS *Inconclusive* cell type replaces these two types, and it represents that there is a portion of the object within the cell, which does not overlap the whole cell. This characteristic allows 3CRS to represent polylines and points in the same way it represents polygons.

Table 1. 3CRS cell types

Cell type	Description
Empty	The cell is not intersected by the object.
Inconclusive	There is a portion of the object within the cell, and it does not fill the cell.
Full	The cell is fully occupied by the object. This type of cell only exists when representing polygons.

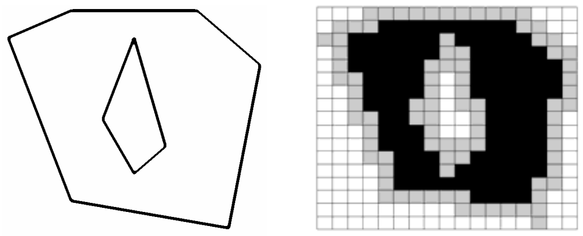


Figure 2. Example of 3CRS representation of a polygon

When computing a 3CRS it is not required to compute the exact polygon area within the cell, as it is done when computing a 4CRS signature. In other words, when generating the signature, we do not need to clip the polygon against the cell, and to compute the area of the clipping region, which is very time-consuming. Instead, it is required only to evaluate if the cell is crossed by the object. As a result, each cell type is computed fast, and objects that do not have an area within the cell, such as points and polylines, can be represented using 3CRS. In the case of points, the 3CRS signature is composed by only one inconclusive cell, while regards to polylines, the 3CRS contains *Empty* cells (cells that are not intersected by the polyline) and *Inconclusive* cells (cells that are crossed by the polyline).

3.2 Algorithm to generate 3CRS

The algorithm to generate 3CRS can be divided in three steps:

1. Compute the MBR-2^n that encloses the object. The MBR-2^n is an expanded MBR where its coordinates are multiples of 2^n, as shown in [15];
2. Follow the object segments, setting as *Inconclusive* the type of the cells they intersect;
3. If the object is a polygon, scan the signature cells, adjusting the type of unmarked cells as *Empty* or *Full*, if the cell is outside the polygon or inside it, respectively. Otherwise, if the object is a polyline or a point, this step is unnecessary, since all unmarked cells are empty.

In the first step, the MBR-2^n of the polygon is computed according to the algorithm presented in [15], and a grid of empty cells is computed from this MBR.

In the second step, the algorithm scans the object's segments. For each segment *s*, it goes from the first cell (where an ending point of *s* is within) to the last cell (where the other ending point of *s* is within) marking the type of these cells as *Inconclusive*. The algorithm goes from one cell to another according the edge of the cell that is intersected by *s* (left, right, bottom or top edge). The cell's edge that is intersected by the segment *s* is identified using the Sutherland-Cohen line clipping algorithm ([11]). The cell adjacent to the current cell's edge is marked as *Inconclusive* and becomes the current cell. The process is repeated until the current cell is equal to the last cell.

It is important to emphasize some aspects about this step:

- There is no specific order to evaluate polygon segments, since all information needed to evaluate one segment is stored within it;
- It is possible to get a cell that is indexed in the signature array in a constant time using a mod operation. Hence one can compute the cell for a point in constant time;
- Given the line equation, one can go along the line and mark intersected cells in a constant time per cell. Thus, for a line segment, the time required will be linear in the number of marked cells;
- Usually a line segment is short and will mark only a constant number of cells, hence requiring O(1) time per line segment;
- Finally, this step computes all partial cells (*Inconclusive* cells), and it needs O(*n*) time if all segments are sufficiently short when compared to the cell size.

During segment evaluation, if the object from which the signature is being computed is a polygon, a matrix of border intersection is used to store a value for each cell, representing a type of intersection between the segment and the left and right borders of the cell. This value is based on the *InsideAbove* flag of the segment. The *InsideAbove* flag of a segment is true when the area inside the polygon lies above the segment; or, if the segment is a vertical line, it indicates that the area inside the polygon is on the left of the segment. The value of 1 is added to the cell value if the *InsideAbove* flag of the segment is true. Otherwise, 1 is deducted from the cell value. Figure 3 shows an example of matrix values. The inconclusive cells of the signature are represented as dark gray cells.

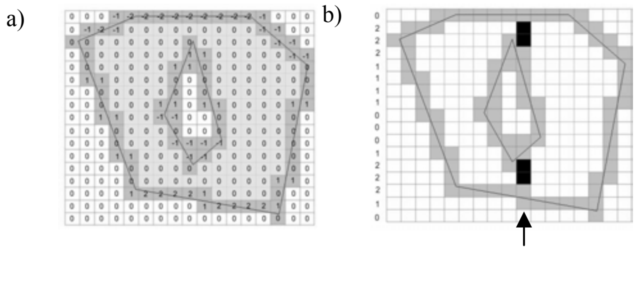


Figure 3 . (a) Matrix of border intersection (b) example of the third step execution

The third step (only used when computing 3CRS from polygons) is responsible for marking the cells that are inside the polygon as *Full* cells. The algorithm goes from bottom to top, following each column of the *matrix*, adding the value of the matrix of border intersection corresponding to each cell. A variable named *counter* is used to store this sum. For each cell whose type was not already set as *Inconclusive*, we evaluate the current value of the *counter* variable. Since the maximum number of cells on the grid is a fixed *K* value, then this step requires O(*K*) time. Figure 3.b shows an example of the third step execution for the column pointed by the arrow. This example uses the matrix presented in Figure 3.a.

Inconclusive cells are already marked (they are represented as the gray cells). On the left of the grid, the numbers correspond to the values of the *counter*, as the algorithm goes through each row. The counter starts with zero. In the first row (from the bottom to the top), the polygon does not intersect the sides of the cell, so the value of counter stays zero. The cell is not marked as *Inconclusive*, so it stays as *Empty*, because the counter is equal to zero. On the second row, the correspondent cell of the *matrix* has the value 1 (as calculated on the previous step of the algorithm, because one segment of the polygon intersects the right side of the cell and the *InsideAbove* flag of that segment is true), so we increase the value of counter by one. The cell was marked before as *Inconclusive*, so its type does not change. The same occurs in the third row. On the fourth row, the value of the counter is two, and the cell is not *Inconclusive*. Hence the cell is marked as *Full*. The algorithm continues to the end of the column (and lines), until all the cells are marked.

3.3 Intersection test using 3CRS

When evaluating two 3CRS signatures of polygons, it is essential that both of them have the same cell size. If it does not apply, it is imperative to perform a change of scale. Whenever a change of scale is necessary, it is accomplished through the grouping of 2^m cells, bearing in mind that the coordinates of the beginning of each cell are proportional to the length of its side. An algorithm for scale change is presented in [15].

After performing the scale changes (if so required), the cells of two 3CRS that overlap each other are evaluated. Only the cells that are inside the intersection of the signatures' MBR are processed. The result of the comparison between two cells is presented in Table 2. Note that there is only one *Inconclusive* result (*Perhaps*), and it occurs when comparing two *Inconclusive* cells.

Table 2. Possible results when comparing two cells

	Empty	**Inconclusive**	**Full**
Empty	No	No	No
Inconclusive	No	Perhaps	Yes
Full	No	Yes	Yes

Figure 4 presents a proposal of algorithm for 3CRS comparison based on the comparison results for the pair of cells presented on Table 2. If all cell comparisons result in "No", then there is no intersection between the polygons. On the other hand, if a "YES" result is found, it means that the polygons intersect, and the comparison can stop. During the comparison, if there is a "PERHAPS" result and no "YES" result, then it is not possible to ensure that the polygons intersect or that they do not has intersection. In this case, it is necessary to execute the refinement step, seeking and reading the polygons' exact representations from disk and executing the exact test.

```
algorithm hasIntersection(signat3CRS1,
                          signat3CRS2)
  interMBR = intersectionMBR(signat3CRS1,
                             signat3CRS2);
  if (signat3CRS1.lengthOfCellSide <
      signat3CRS2.lengthOfCellSide)
    s3CRS = changeScale(signat3CRS1,
            signat3CRS2.lengthOfCellSide);
    b3CRS = signat3CRS2;
  else
    if (signat3CRS1.lengthOfCellSide >
        signat3CRS2.lengthOfCellSide)
      b3CRS = signat3CRS1;
      s3CRS = changeScale (signat3CRS2,
              signat3CRS1.lengthOfCellSide);
    else
      s3CRS = signat3CRS1;
      b3CRS = signat3CRS2;
  result = NO;
  for each b3CRS cell b that is inside
                              interMBR do
     for each s3CRS cell s that intersects
                                cell b do
       if b.type == EMPTY or s.type == EMPTY
          continue;
       if b.type == INCONCLUSIVE or
          s.type == INCONCLUSIVE
          result = PERHAPS;
       if ( (b.type == FULL) and
            (s.type == FULL or
             s.type == INCONCLUSIVE) ) or
          ( (s.type == FULL) and
            (b.type == FULL or
             b.type == INCONCLUSIVE) )
          return YES;
  return result;
```

Figure 4. Algorithm for 3CRS comparison

4. EVALUATION TESTS

This section is dedicated to presenting the experimental results concerning the evaluation of the use of the 3CRS signature in query processing. We evaluated the use of 3CRS as a filter in the second step of MSQP ([6]) against the use of 4CRS, and against the processing without a filter step, the architecture of two steps ([10] and [12]). We evaluated the intersection join of set of polygons.

4.1 Datasets

The polygon real datasets used in the experiments consist of township boundaries, census block-group, geological map and hydrological map of Iowa (US), available online from "http://www.igsb.uiowa.edu/nrgis/gishome.htm", and Brazilian municipalities (IBGE, 1996). In order to simulate large datasets, the Iowa datasets were replicated six times, in the same way as suggested by [6]. The original polygons were shifted by random displacements of *x* and *y* coordinates. In the case of the Brazilian municipalities, we performed one replication (named Brazilian municipalities'), so that we could execute the test of Brazilian municipalities against Brazilian municipalities'. Data characteristics are presented in Table 3. The test datasets are composed of objects of medium complexity - less than 1000 segments per object. The more complex is the object, the greater is the time spent in the plane sweep algorithm (3rd step). In this way, we can expect more performance gains when datasets composed of more complex objects are used.

Table 3. Test datasets

Datasets		Size (KB)	# pol.	# segments	Avg. # segm.
Iowa	Census block group	38,824	17,844	1,764,588	98
	Topography	61,748	20,070	3,780,552	188
	Hydrologic map	6,904	2,544	475,434	186
	Township boundaries	25,288	12,216	1,059,438	86
	Geological maps	21,856	9,984	640,428	64
Brazil	Municipalities	9,840	4,645	399,002	85
	Municipalities'	9,840	4,645	399,002	85
Average		24,757	10,278	1,216,921	118

4.2 Test Environment and R*-tree characteristics

Tests were executed on a PC powered by an Athlon XP 1600+ 1.4 GHz CPU with 256MB RAM. A page size of 2,048 bytes for I/O operations was defined.

The R*-tree ([2]) was chosen as a spatial access method in order to reduce the search space. In other words, the R*-Tree was used to only take into account objects that have at least a MBR intersection and not all of them. That choice was due to the wide use of R*-Tree as well as to the successful results found in the literature. The access methods traditionally used employ the object's Minimum Bounding Rectangle (MBR), and the access methods execution returns what is called a set of candidates, since it contains all pairs of polygons that belong to the answer plus other pairs that have only a MBR intersection. In the same way as [6] and [15] do, for our tests we generated R*-Trees that store the 4CRS signatures as part of the polygons' keys. This means that they were stored in the leaf nodes of the R*-Tree index.

The tests (Table 4) can be described according to the concepts presented in Sub-Section 2.1 (Architectures for Processing Spatial Joins). The experimental tests using 3CRS and 4CRS were executed according to the MSQP (architecture of three steps), while the tests without using a filter step can be described as the architecture of two steps. In all experiments we have shown the average results of these four joins, except for signatures characteristics (Table 5).

Table 4. Joins executed to test the algorithm that computes the approximate area of polygon x polygon intersection

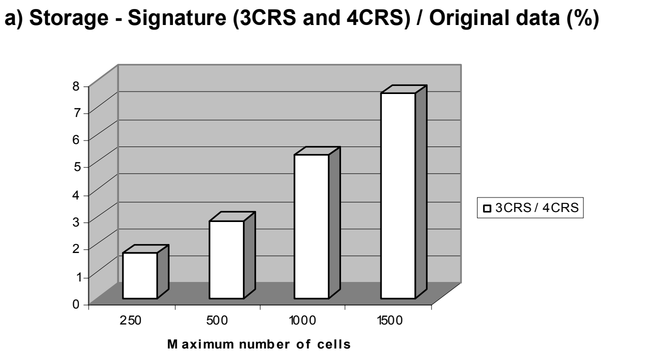
Labels	Dataset 1	Dataset 2
Join-1	Geological map	Township boundaries
Join-2	Geological map	Census block
Join-3	Township boundaries	Census block
Join-4	Brazilian municipalities	Brazilian municipalities'

4.3 Experimental Results

In order to generate the raster signatures, we have to choose the maximum number of grid cells ([15]). Intuitively, the larger the number of cells, the closer the approximation to the original polygon is. However, processing large size raster signatures could produce high I/O and CPU costs. To evaluate the effects of the different choices, we executed experimental tests using different maximum numbers of cells, such as: 250, 500, 1,000 and 1,500. We executed the following evaluations: storage requirements; number of pairs identified in the second step which represent hits (pairs of objects that intersect each other) and false hits (pairs of objects that do not have intersection) - those are identified without executing the refinement step; CPU costs; and, I/O costs.

Despite of the 3CRS being based on the 4CRS, it presents some performance differences compared to 4CRS, in terms of generation time, execution time and number of inconclusive pairs (pairs of objects that are not identified in the sencond step as hits or false hits, and which must be processed in third step - Figure 1.b - Section 2).

As regards storage requirements, first, it is important to emphasize that signatures with 3 or 4 colors (3CRS or 4CRS) have the same storage requirements, since 2 bits are required to store the color of each cell. Therefore, both of them use the same space to be stored. Figure 5.a presents the size of the signature in comparison with the size of the original dataset. Signatures of maximum number of cells equal to 1,500 have more storage requirements.



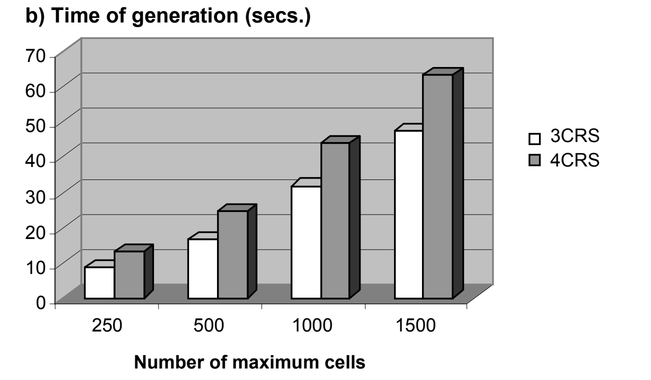


Figure 5. a) Size of the signatures (3CRS and 4CRS) related to the real data. b) Generation time for 3CRS and 4CRS

In relation to generation time, 3CRS can be generated faster than 4CRS signatures, as shown in Figure 5.b. As expected, when the maximum number of cells increases, the time to generate the signatures also increases.

Table 5 presents 4CRS and 3CRS signatures characteristics when the maximum number of cells equals 500. To store 4CRS or 3CRS signatures when the maximum number of cells equals 500 needs only, on average, 2.85% of the space required to store the real datasets. 3CRS can be generated in approximated 70% of 4CRS generation time.

Signatures of 250 maximum number of cells are processed faster, since there are less cells to process, while signatures of 1,500 maximum cells are processed slower, since there are more cells to evaluate. Figure 6 shows the average time to execute the first two steps of the architecture, using 3CRS and 4CRS. In both cases, the bigger the maximum number of cells, the bigger the processing time is.

Table 5. Raster signatures' characteristics with maximum number of cells equal to 500 for each dataset

Datasets		Data size (KB)	4CRS size (KB)	4CRS / data (%)	Gener. time 3CRS (sec.)	Gener. time 4CRS (sec.)	3CRS / 4CRS (%)
Iowa	Census block group	38,824	1163	3.00	14.83	20.77	71.40
	Hydrological map	6,904	169	2.45	2.70	3.97	68.02
	Topography map	61,748	1455	2.36	33.71	50.67	66.54
	Township boundaries	25,288	838	3.31	10.18	14.53	70.09
	Geological maps	21,856	676	3.09	8.43	11.93	70.64
Average		29.050	827	2.85	13.97	20.37	69.34

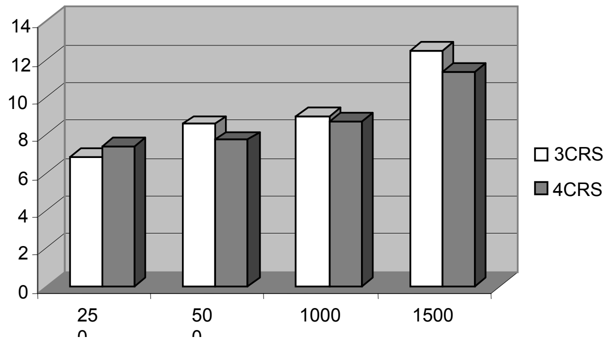


Figure 6. Average time (in seconds) to execute the first two steps of the signature, using 3CRS and 4CRS

The number of inconclusive pairs (or candidate pairs - Figure 1.b - Section 2) that go from the second step to the third step is bigger when 250 is chosen as the maximum number of cells, as demonstrated in Figure 7, Figure 8 and Table 6. In other words, signatures with a maximum number of cells equal to 250 identify less hits and false hits. On the other hand, when the maximum number of cells is 1500, the number of inconclusive pairs is the smallest. The 3CRS and the 4CRS have almost the same results. 3CRS generates only a few more inconclusive pairs to the third step (Figure 7). It is important to emphasize that comparing the three -step processing against the two step processing, the reduction of number of objects that are processed by the exact geometry test is around 75% (78%) to 89% (91%) when using 3CRS or 4CRS respectively (Figure 8).

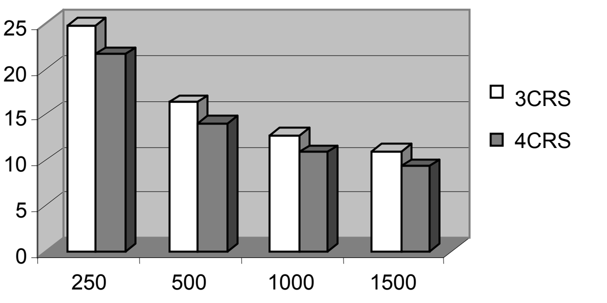


Figure 7. Inconclusive pairs (%) that are processed by the third step

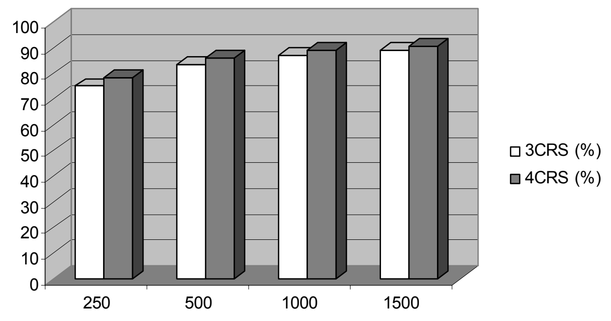


Figure 8. Hits and false hits pairs (%) identified by the filter step (3CRS and 4CRS)

Table 6. Percentage of inconclusive pairs that go from the second to the third step

Maximum number of cells	3CRS (%)		4CRS (%)	
	Hits and false hits	Inconclusive	Hits and false hits	Inconclu -sive
250	75.16	24.84	78.38	21.62
500	83.62	16.38	85.98	14.02
1000	87.27	12.73	89.17	10.83
1500	89.09	10.91	90.69	9.31

Despite the differences between 3CRS and 4CRS, both present significant gain over the 2-step architecture, in terms of total execution time and disk accesses.

Total execution time is presented in Table 7 and Figure 9. Notice that both 3CRS and 4CRS use much less time than the 2-step architecture (over 50% reduction). The time used in 3CRS is slightly bigger than 4CRS, because the 3CRS test generates more inconclusive pairs, which leads to more exact and slower tests.

Table 7. Total execution time (secs.) between 3CRS, 4CRS and 2-step architecture

Maximum number of cells	3CRS	4CRS	2-step
250	37.47	23.53	58.40
500	30.12	30.48	66.10
1000	25.00	25.30	63.39
1500	30.05	23.94	69.19

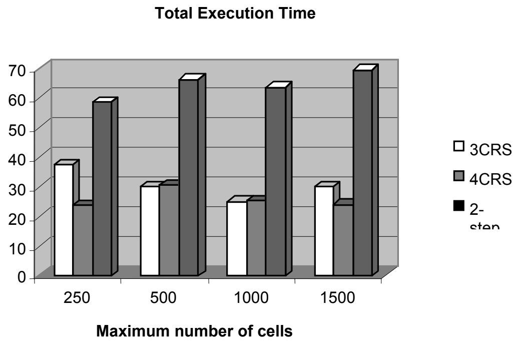


Figure 9. Comparison of total execution time (secs.) between 3CRS, 4CRS and 2-step architecture

Table 8 and Figure 10 show the relation between disk accesses using the 3-step architecture (3CRS and 4CRS) and 2-step architectures. It shows the ratio in percentage: number of disk accesses used by the 3-step architecture divided by number of disk accesses used by the 2-step architecture. Notice that 4CRS uses slightly less disk accesses than the 3CRS. Again, it is because 3CRS generates more inconclusive pairs, which leads to a more exact test (the refinement step). Therefore, the algorithm needs to access the disk more often, to seek and read objects from disk and execute the exact test. However, it is important to emphasize that the difference between the number of disk accesses using 3CRS and 4CRS is very small.

Table 8. Disk accesses (average) of the 3-step join over the 2-step join

Maximum number of cells	3CRS	4CRS
250	52.11	48.50
500	51.00	48.18
1000	59.64	57.41
1500	71.07	69.13

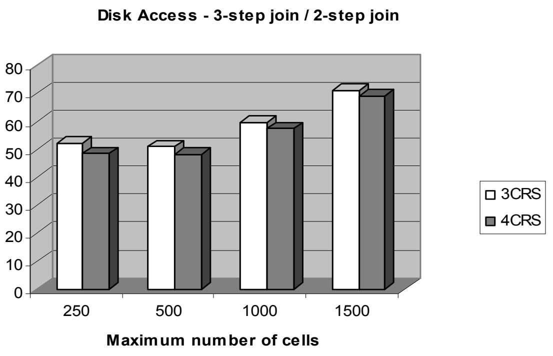


Figure 10. Comparison of the disk accesses (average) of the 3-step join over the 2-step join

5. CONCLUSION

Spatial operations are very costly. The literature presents different approaches for processing spatial operations. This work proposed a new raster signature, the 3CRS, based on the 4CRS ([15]), that can be used as a second filter step in the Multi-Step Query Processor of [6]. 3CRS has the good performance of the 4CRS when compared against the 2-step processing. Moreover, 3CRS has faster generation time over 4CRS and it is also more flexible, since it can be used to represent different spatial data types, such as polygons, polylines and points. Due to its faster generation time, 3CRS can also be calculated on-the-fly, for example, when the optimizer decides to use it in case of intermediate results of multi-joins. For example, the average time spent to perform the experiments using 500 cells (30.48 seconds - Table 7) plus the time required to generate the signatures (13.97 seconds - Table 5) is less than the time spent to perform the query using only 2-steps (66.10 seconds - Table 7). This makes the 3CRS approach a better choice to be implemented in core spatial databases than 4CRS. Another advantage is that the intersection of two 3CRS (as a result of a previous operation, for example in a multi-join) can be computed using only the 3CRS signatures themselves: it is not necessary to retrieve the object from disk. We only have to compute empty, inconclusive and full cells to find the resulting 3CRS.

The experiments we executed to evaluate the new signature demonstrated the effectiveness of 3CRS. It only needed, on average, 2.85% of the storage requirement to store 3CRS related to the real datasets. It showed a significant reduction in generation time (30% reduction, on average) in relation to 4CRS. The time and number of disk accesses to process the queries where much smaller than the time to execute the queries without the signature. The filter step identifies more than 75% of the candidate pairs, which are not sent to the exact geometry test. In relation to 4CRS, the processing using 3CRS produces a small growth of inconclusive answers. In other words, the increase of the total processing time/number of disk accesses is very small, and encourages the use of 3CRS in processing spatial joins.

As future works, we intend to develop a different storage mechanism to reduce storage requirements. We also plan to evaluate the use of 3CRS to representing polylines and points. Another future work is to implement this signature in SECONDO ([9]).

6. REFERENCES

[1] Aronoff, S. *Geographic Information Systems*, 1 ed., WDL Publications, Ottawa, Canada, 1989.

[2] Beckmann, N., Kriegel, H. P., Schneider, R., and Seeger, B. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data* (Atlantic City, NJ, USA, May 23-25, 1990), ACM Press, New York, NY, 1990, 322-331.

[3] Boissonnat, J. D., and Preparata, F. P. Robust Plane Sweep for Intersecting Segments, SIAM Journal on Computing, 1997, v. 29, issue 5, 1401-1421.

[4] Brazilian Institute of Geography and Statistics Fundação Instituto Brasileiro de Geografia e Estatística – IBGE: "Malha Municipal Digital do Brasil - 1994", Rio de Janeiro, 1996.

[5] Brinkhoff, T., Kriegel, H. P., and Schneider, R. Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems. In *Proceedings of Ninth International Conference on Data Engineering (ICDE'93)* (Vienna, Austria, April 19-23, 1993), IEEE Computer Society, Washington, DC, USA, 1993, 40-49

[6] Brinkhoff, T., Kriegel, H. P., Schneider, R., and Seeger, B. Multi-step Processing of Spatial Joins. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data* (Minneapolis, Minneapolis, Minnesota, USA, May 24-27, 1994) ACM Press New York, NY, USA, 1994, 197-208.

[7] Freiseisen W., and Pau, P. A generic plane-sweep for intersecting line segments, Technical Report RISC-Linz TR-98-18, University of Linz, Linz, Austria, 1998.

[8] Gordon, S. R.,Goodwin, C.W.H., and Xiong, D. Final Report on Status of Spatial/Map Databases. Technical Report of Oak Ridge National Laboratory, June, 1994.

[9] Güting, R.H., Almeida, V., Ansorge, D., Behr, T., Ding, Z., Höse, T., Hoffmann, F., Spiekermann, M., and Telle, U. Secondo: An Extensible DBMS Platform for Research Prototyping and Teaching, In *Proceedings of 21st International Conference. on Data Engineering (ICDE'05)* (Tokyo, Japan, April 5-8, 2005), IEEE Computer Society Washington, DC, USA, 1115-1116.

[10] Kothuri, R. K., and Ravada, S. Efficient Processing of Large Spatial Queries Using Interior Approximations. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD'01)*, (Redondo Beach, CA, USA, July 12-15, 2001) Springer-Verlag London, UK, 2001, 404-424.

[11] Newman, W. M., and Sproull, R. F. Principles of Interactive Computer Graphics, 2 ed., McGraw-Hill Book Company, New York, 1979.

[12] Orenstein, J. A. Spatial query processing in an object-oriented database system. In *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, (Washington, DC, USA, May 28-30, 1986) ACM Press, New York, NY, USA, 1986, 326-336.

[13] Samet, H.The Design and Analysis of Spatial Data Structure, 1 ed., Addison-Wesley Publishing Company, Boston, Massachusetts, 1990.

[14] Tao, Y., Sun, J., and Papadias, D. Selectivity estimation for predictive spatio-temporal queries. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)* (Bangalore, India, March 5-8, 2003) IEEE Computer Society, 2003, 417-428.

[15] Zimbrao, G., and Souza, J. M. A Raster Approximation For Processing of Spatial Joins. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB'98)* (New York City, NY, USA, August 24-27, 1998) Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 558-569.

[16] Zhu, H., Su, J., and Ibarra, O. H. Toward Spatial Joins for Polygons. In *Proceedings of the 12th International Conference on Scientific and Statistical Database Management (SSDBM'00)* (Berlin, Germany, July 26-28, 2000) IEEE Computer Society Washington, DC, USA, 2000, 231-244.