Vicinities for Spatial Data Processing: a Statistical Approach to Algorithm Design

Peter Y. Wu
wu@rmu.edu
Department of Computer and Information Systems

Sushil Acharya
acharya@rmu.edu
Department of Engineering

Robert Morris University
Pittsburgh, PA 15108, USA

Abstract

Spatial data processing is often the core function in many information system applications. Algorithm design for these applications generally aims at being worst case optimal for processing efficiency. We propose a different approach applying the notion of vicinity. We partition the object space into grid cells of size adapted to the statistical dimensions of the input data objects for processing, and consider only those data objects sharing the same common grid cells. We describe the processing steps of the algorithm in our approach and analyze the performance. We also experimented with different data patterns in our implementation. We believe that our approach can be efficient and practicable for the computation of geometric intersection and spatial interference detection. These are essentially the core functions in geographic information systems, computer graphics and computer aided design systems as well. We also briefly discuss our understanding of how the grid cell size may affect the performance with regard to varying patterns of the input data objects.

Keywords: vicinity, spatial data processing, algorithm design, algorithm analysis.

1. INTRODUCTION

Information systems quite often need to deal with spatial data. This can be map data in geographic information systems, or architectural blueprint or circuit schematic in computer aided design systems, or graphic objects in robotics simulation and computer vision systems. These applications need as a core function spatial data processing that is efficient for the system to be practicable (Sutherland, Sproul & Schumacker, 1972; Foley, van Dam, Feiner, Hughes & Phillips, 1994).

Algorithm design for these applications by practitioners in computer science traditionally depends on the analysis techniques based on a worst case optimal strategy. Developed from the analysis of sorting algorithms (Knuth 1972), the optimal performance when dealing with problem size of N is well established to be of the order N*log(N) (Aho, Hopcroft & Ullman, 1974). When handling multiple data sets in spatial data processing, such as interference detection of geometric objects in robotics, or map overlay in geographic information systems, the optimal algorithm to compare two data sets of sizes M and N is of order M*log(M)+N*log(N)+K where K is the number of intersection or interference

points (Nievergelt & Preparata, 1982). We note that the complexity analysis is almost always done for the worst case scenario. While these research results often reveal much about the nature of the problem, they fall short being practicable and applicable from an information system standpoint (Preparata & Shamos, 1986).

We propose a different approach in this paper. Instead of complexity analysis for the worst case scenario, we apply the notion of vicinity in the algorithm design. In our approach, we create vicinities around the data objects and observe that interference between these objects must occur in the same vicinity. Therefore, we only need to process those objects sharing in the same vicinity. The idea was first applied to solve the polyline intersection problem using adaptive grids (Franklin, 1983), and to spatial interference detection between geometric objects in computer graphics and computer aided design (Franklin, Chandrasekhar, Kankanhalli, Akman & Wu, 1990). It was also extended to compute intersection between line segments in geographic information system for map overlay (Wu & Franklin, 1990; Wu, 2005). This paper presents the approach in a more generalized setting of spatial data processing for discussion.

Sections 2 and 3 will describe how we create a structure to apply the notion of vicinity to the processing of spatial data. Section 4 presents the algorithm in processing steps and analyzes the time complexity of each step to assess the overall performance. Section 5 will then discuss an interesting parameter – grid cell size for the spatial structure to foster the notion of vicinity. The parameter affects efficiency but we may find a value range for which system performance is reasonably stable. Section 6 gives a summary in conclusion.

2. THE NOTION OF VICINITY

The notion of vicinity refers to our understanding that when objects interfere with one another, they are in the same vicinity. These objects may be simple line segments, such as in the case of map overlay in geographic information systems; these are fundamental units making up the polyline. In the case of spatial interference in robotics, the objects may be fundamental 3D surface patches. To apply the notion to spatial data processing, we impose a cellular partition onto the object space into grid cells. A simple preprocessing step can identify the grid cells an object occupies. If two objects do not share any common grid cell, we know that the two objects do not intersect with each other. Therefore, we need to only examine possible spatial interference between the occupants in each grid cell.

How can we create these grid cells? The grid cells must support the simple preprocessing step in such a way that the step can be completed in linear time. By examining each object once, we must be able to identify all the grid cells it occupies by direct computation. We have hence chosen to use rectangular grid cells of regular orientation, imposed onto the object space to foster the notion of vicinity.

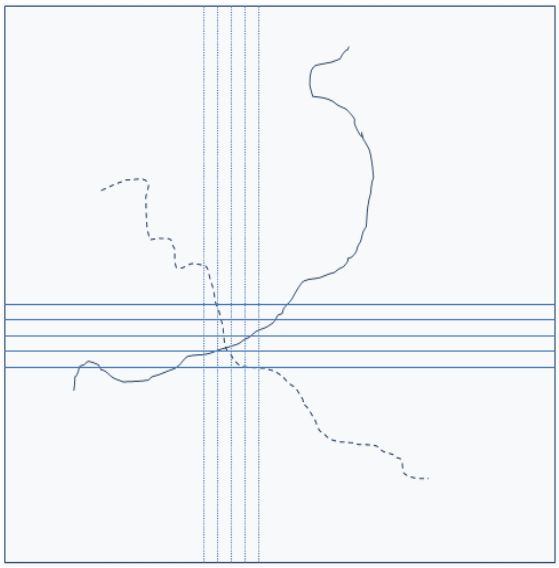

Figure 1. *The grid isolates intersection between two polylines to the constituent line segments in the same vicinity.*

Since the grid cells are regular and rectangular, the preprocessing step is simple computation: for each object, we can determine and report the grid cells it occupies in time linear to the number of grid cells occupied, for 2D as well as 3D cases (Foley & van Dam, 1994). (In fact, the approach also works for higher dimension data sets.) Figure 1 illustrates how the rectangular grid cells isolate the two intersecting line segments in determining the intersection between two polylines in 2D, applying the notion of vicinity.

With the grid cells pattern superimposed over the object space, we apply the notion of vicinity: we only need to look into those grid cells which contain edge segments from both polylines. Each of these grids therefore identifies for us the

potentially intersecting edge segments to be considered. Compared to the naïve method of checking each edge segment from one polyline against those of the other polyline, the number of potentially intersecting edge segments identified by the grid cells can be much reduced.

What then should be the appropriate dimensions for these grid cells? Intuitively, smaller grid cells will be able to better isolate each case of intersection, but the total number of grid cells, and thus the overhead cost of processing will then be higher. Section 5 will study the issue, but in the next section, we will discuss the data structure design to control the overhead cost.

3. GRID CELLS STRUCTURE DESIGN

To preprocess the data, we need to examine each object to determine the grid cells occupied. Given the data object, it takes constant time to compute whether or not it occupies a specific grid cell (Preparata & Shamos, 1986). The time for the preprocessing step is therefore linear to the total number of (object, grid cell) pairs. However, we have to be more careful with the structure design to handle the information for the (object, grid cell) pairs.

For regular grid cells in a rectangular pattern, in 2D, 3D or higher dimensions, we can effectively index through each grid cell by its subscripts as coordinates. Thus, we can calculate the grid cell index subscripts from its geometry – its location and dimensions, and vice versa. It allows us to implement a hash coding function to search into the data structure for the grid cells, fetching the data objects occupying each cell (Knuth, 1972, pp.506-542). Hence, when preprocessing the data objects, we can treat each grid cell as a bucket to receive the identity of the object if the object occupies that grid cell. More, we will be able to allow empty grid cells to be non-existent in the data structure, taking up no storage on the one hand, but also requiring no time to examine them on the other. This type of associative storage scheme is in use in many existing software systems, such as the cache in M Programming (Walters, 1997), and in Prolog facts storage and many deductive databases (Clocksin & Mellish, 2003; Krikelis & Weems, 1997; Colomb, 1998).

4. THE ALGORITHM AND ITS ANALYSIS

With the grid cells structure design in place, we now go on to describe the processing steps of the algorithm below. For detection of general spatial interference in one set of objects, the algorithm goes through the following steps.

1. Set up the grid cells structure.
2. Determine for each data object the grid cells it occupies.
3. For each non-empty grid cell, determine spatial interference between objects in that cell.

Step 1 sets up the grid cell structure. There is an interesting issue about the appropriate grid cell dimensions. We will discuss that in the next section. But in this step, we first evaluate the statistical measures of our input data objects. Our general approach is to take the dimensions of the bounding rectangle for each object, and use that to find the average size as appropriate for our grid cells (2D as well as 3D).

This statistics evaluation should be a constant time computation for each object, as we may understanding from computational geometry (Preparata & Shamos, 1986). The total time complexity therefore should be linear to the number of data objects, noted as order(N), N being the number of objects.

Step 2 is the preprocessing step of determining the grid cells each object occupies, referred to in Section 2. With the dimensions of the bounding rectangle, we can directly derive the grid cells occupied. Using the content associative scheme of storage structure (Krikelis & Weems, 1997), the index to each grid cell also serves as the access point to collect the (object, grid cell) pairs information in the storage structure.

This preprocessing step requires us to examine each data object, but for each object, we also need to report and collect information for each grid cell it occupies. The time complexity will be of order(N*M), N being the number of objects and M the number of grid cells the object occupies. Note that M is related to grid cell size used in Step 1, but M is independent of N. If object sizes are relatively the same, M generally stays constant.

Step 3 is the interference detection step. Since the complexity to determine interference between two fundamental objects is a constant measure, we will have to count the number of times we compare objects one to another. With what we have constructed in Step 2, we now only need to iterate through each non-empty grid cell. If the cell contains more than one data

object, we will examine one against another for spatial interference. There are a few points to note. First, we expect the number of objects to have been reduced when the objects under consideration do not share vicinities in common. Second, we may still apply a sophisticated sorting algorithm to avoid naively doing pairwise comparison between the objects. We avoid the complexity of order(N^2), but apply that of order N*log(N), for N being the number of data objects in the cell (Lee & Preparata, 1984). Third, objects which interfere each other may occupy more than one common grid cell. We can keep a record of the pairs we have processed already to avoid re-doing the pairs we have already processed.

Granted that we can set up our system to do the processing intelligently, we still achieve the time complexity of the chosen algorithm for spatial interference detection, which is still of order N*log(N) in the worst case. But if the input data happens to allow the grid cells pattern to sort out the non-interfering cases when the objects under consideration do not share vicinities in common, we have greatly improved the overall efficiency of the system.

5. GRID CELL SIZE

To set up the grid cells structure, we need to determine an appropriate size for the grid cells. Since the grid is intended to isolate cases of spatial interference between objects, it may seem desirable to use smaller grid cells so that two different objects may not occupy a common grid cell unless they interfere with each other. On the other hand, small grid cells result in a grid with more cells, and a larger M in Step 2.

Let T_1 be the time for Step 1 to set up the grid. T_1 is proportional to the size of input data set, and therefore constant for any given data set. Let T_2 be the time needed for Step 2, the preprocessing step, and T_3 the time for Step 3, the interference detecting step. If A is the measure of the size of a grid cell, it is therefore inversely proportional to the number of cells in the grid for a given input data set. Then we have

$$T_2 = \text{Order of } (1/A).$$

$$T_3 = \text{Order of } (A^2).$$

Since T_2 is generally monotonically decreasing and concave downwards while T_3 is generally monotonically increasing and concave upwards, the total time $T=T_1+T_2+T_3$ would generally attain minimum at A' for which $T_2= T_3$. Around A=A', the total time T also shows a region of minimal curvature. T will be relatively stable for variations of A. We illustrate this in Figure 2.

For Step 1, when we survey the statistical characteristics of the input data, and have opted to apply the average object size to be the grid cell size for A, we find the total computation time T attains minimum and remains relatively stable to changes in A. In other words, we have already arrived at A=A'. But when the variance in the object sizes of the input data is large – while the average grid cell size remains unchanged, the total time T becomes much more sensitive to changes in A. Our intuitive understanding is the following: when the grid cell size is about the same as the object sizes, our approach captures well the notion of vicinity in processing, thus attaining the minimal T there. But when the object sizes vary as indicated by a larger value of the variance, the grid does not work so well to capture the vicinity of the data objects, resulting in the degradation of its performance, and a larger value for T.

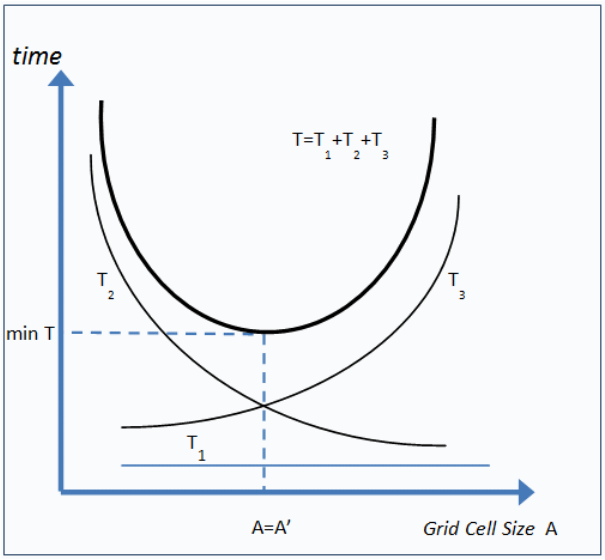

Figure 2. *Different Grid Cell sizes result in changes in time performance of the algorithm.*

6. SUMMARY

We presented an approach to algorithm design for spatial data processing which is different for the traditional way by practitioners in computer science. Instead of seeking to be worst case optimal, we apply an intuitive understanding from the notion of vicinity. We impose a grid pattern over the object space and use the grid cells occupied by each data object as a measure

of its vicinity. We observe that when two objects interfere with each other, they must occupy some common grid cell. Therefore, only the grid cells occupied by two or more potentially interfering objects need to be examined. Applying the same interference detection algorithm only to the cases in each such grid cell, we solve the same spatial interference detection problem, possibly more efficiently. We also argue that at worst, it would be the same complexity as the interference detection algorithm employed. An interesting issue in the whole approach is the appropriate grid cell size to set up the grid. In our discussion, we suggest that the average dimension of the objects would be an appropriate measure for the grid cell size, and that the performance is not particularly sensitive to the choice around that proposed value.

7. REFERENCES

Aho, A.V., J. Hopcroft & J.D. Ullman. (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley.

Clocksin, W.F. & C.S. Mellish. (2003). *Programming in Prolog: Using the ISO Standard*, 5th edition, Springer.

Colomb, R.M. (1998). *Deductive Databases and Their Applications*, Taylor and Francis.

Franklin, W.R. (1983). Adaptive Grids for Geometric Operations*, 6th International Symposium on Automated Cartography*, (*AUTO-CARTO 6*), *Vol.2*, pp.230-239.

Franklin, W.R., N. Chandrasekhar, M. Kankanhalli, V. Akman & P.Y. Wu. (1990). Efficient Geometry Operations for CAD, *Geometric Modeling for Product Engineering*, Wozny, Turner & Preiss (eds), Elsevier Science B.V. (North-Holland), pp.485-498.

Foley, J.D., A. van Dam, S.K. Feiner, J.F. Hughes & R.L. Phillips. (1994). *Fundamentals of Computer Graphics*, Addison-Wesley.

Goodrich, M.T. & R. Tamassia. (2006). *Data Structures and Algorithms in Java* (4th ed) Wiley.

Krikelis, A. & C.C. Weems. (1997). *Associative Processing and Processors*, IEEE Computer Science Press.

Knuth, D.E. (1972). Sorting and Searching, *The Art of Computer Programming, Vol.3*, Addison-Wesley.

Lee, D.T. & F.P. Preparata. (1984). Computational Geometry – A Survey, *IEEE Transactions on Computers, C-33(12),* pp.1072-1101.

Nievergelt, J. & F.P. Preparata. (1982). Plane-Sweep Algorithms for Intersecting Geometric Figures, *Communications of ACM*, 25(10), pp.739-747.

Preparata, F.P. & M.I. Shamos. (1986). *Computational Geometry: an Introduction*, Springer-Verlag.

Sutherland, I.E., R.F. Sproul & R.A. Schumacher. (1974). A Characterization of Ten Hidden-Surface Algorithms, *ACM Computing Surveys, 6(1)*, pp.1-55.

Walters, R.F. (1997). *M Programming: A Comprehensive Guide*, Elsevier, Oxford, U.K.

Wu, P.Y. & W.R. Franklin. (1990). A Logic Programming Approach to Cartographic Map Overlay, *Computational Intelligence Journal, 6(2),* National Research Council of Canada, May 1990, pp.61-70.

Wu, P.Y. (2005). A Distributed Approach to Fast Polygon *Overlay, 6th Annual Central Appalachian Geo-Spatial Conference*, California University of Pennsylvania.