Fine-Grained Quadtree-based Polygon Indexing in CudaGIS

(Initial Design & Implementation)

Jianting Zhang[1, 2] and Simin You[1]

1 Department of Computer Science, the City College of the City University of New York
2 Dept. of Computer Science, the Graduate Center of the City University of New York

Different from indexing MBRs of polygons using quadtree/R-Trees [1] (Oracle Spatial) or generalized indexing search trees [2] (PostgreSQL/PostGIS), Microsoft SQLServer Spatial decomposes polygons into four levels of grid cells with adjustable cell sizes so that these grid cells can be indexed by B+-Tree for query processing [3] as illustrated in the left side of Fig. 1. Indexing polygons directly through decompositions is computationally intensive and it is desirable to use GPU accelerations. Different from using a fixed number of grid levels as used in SQLServer Spatial, our idea is to construct quadtrees on polygons by examining spatial relationships (outside, intersect and inside) between quadrants under a quadtree node with a polygon in parallel and construct child nodes based on the resulting spatial relationships (white, gray and black nodes, respectively) and expand the quadrants that intersect with the polygon as necessary level by level as illustrated in the right side of Fig. 1. We next discuss the criteria in determining the spatial relationships and the mapping between the level-wise construction algorithm and the GPU hardware by exploiting fine-grained parallelism.

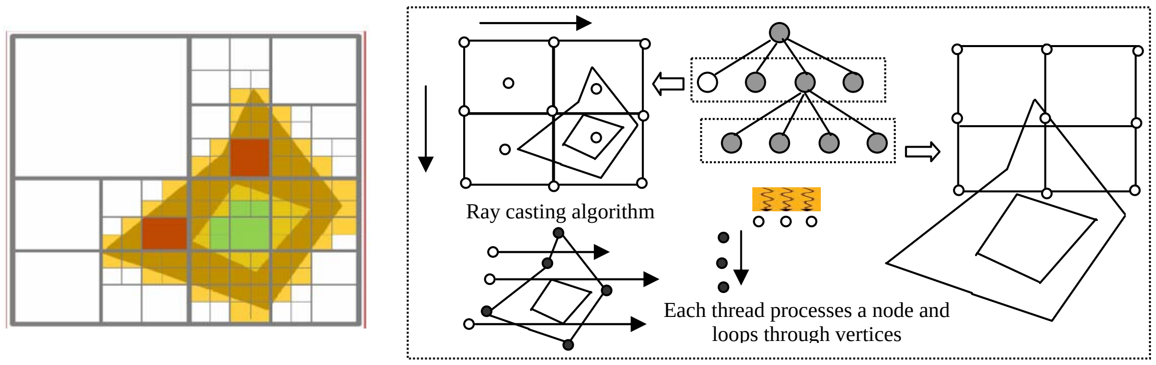

Fig. 1 Parallel Indexing on Polygon Internals

Three sets of tests are needed to determine the spatial relationship for a quadrant. Several well-established computational geometry principles can be used to test the relationships between a rectangle (quadrant) and a polygon. Our design adopts the procedure used in [4] for this purpose. First, the polygon's edges are tested whether they intersect the quadrant (Line-Box-Intersection or LBI tests). Second, each of the polygon's vertices is tested on whether they are within the quadrant (Point-In-Box or PIB tests). Third, the center of the quadrant is tested whether it is within the polygons (Point-In-Polygon or PIP tests). The tests tell the following three conditions (1) none of the quadrant's four edges cross through the polygon's edge (2) none of the polygon's vertices are inside of the quadrant (3) the center of the quadrant lies within the polygon. The quadrant is outside of the polygon if (1) and (2) are true but (3) is false and hence a white node is created for the quadrant. The quadrant is inside of the polygon if all the three conditions are true and hence a black node is created for the quadrant. If either (1) or (2) is false, the quadrant intersect with the polygon.

By observing that quite some tests are shared by neighboring quadrants, we propose a design to minimize duplicated computation and improve system performance. Assuming that a polygon with m vertices is assigned to a thread block, n*n quadrants are batch processed by the block as following. First, each of the m-1 edges is sequentially tested against the 2*n(n+1) unique edges in the n*n quadrants in parallel for LBI tests. Second, each of the m vertices is sequentially tested against the n*n quadrants in parallel for PIB tests. Third, each of the m-1 edges sequentially participate the PIP tests for the n*n quadrants in parallel. In all of the three steps, the thread level parallelism is mapped to quadrants. The design utilizes GPU's SIMD feature very well as all threads access the same polygon vertex during a single looping step and memory accesses are perfectly coalesced. Furthermore, neither of the tests incurs divergence. While LBI and PIB tests involves mostly integer comparisons, PIP tests require floating point computation and can utilize GPU's floating point computing power effectively. Choosing the proper value of n, which should be the power of 2, is critical in achieving good performance. If n is too small (e.g., 2), the thread level parallelism can not be fully utilized and the processors within a warp might be idle for this reason. If n is too big, the parallel computing power could be wasted on uniform quadrants that do not need to be sub-divided if a smaller n had been used. We plan to develop optimization strategies to suggest n values based on some easy-to-compute indices of polygons (e.g., height/width ratio).

Reference:

1. R. K. V. Kothuri, S. Ravada and D. Abugov (2002). Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data. Proceedings of the International conference on Management of Data, SIGMOD, 546-557.
2. http://en.wikipedia.org/wiki/GiST
3. Y. Fang, M. Friedman, G. Nair, et al. (2008). Spatial indexing in microsoft SQL server 2008. Proceedings of the International conference on Management of Data, SIGMOD, 1207-1216.
4. K. Wang, Y. Huai, R. Lee, et al. (2012). Accelerating pathology image data cross-comparison on CPU-GPU hybrid systems. Proceedings of the International Conference on Very Large Data Bases, VLDB, 1543-1554