Polygon Overlay Operations in CudaGIS

(Initial Design & Implementation)

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

Polygon overlay operations are important components in vector GIS. Unfortunately, the current implementation of CudaGIS does not include a fully-fledged module to support such operations due to their complexities. However, by leveraging the point-in-polygon test operation that is currently supported by CudaGIS, we have successfully implemented and tested the intersection operation although the design and the implementation still need to be enhanced to handle more special cases. Nevertheless, we would like to share some design and implementation details and encourage interested readers to join our development efforts.

The design is based on the popular polygon clipping algorithm by Greiner and Hormann [1] where three phases are required: testing intersection, determining entry/exit status and tracing output polygons. The algorithm was originally designed for serial implementation where pointer-based linked lists were used extensively for random data accesses which are not suitable for GPUs. As the first two phases are computationally intensive and dominate the whole process, we have decided to implement the first two phases on GPUs. The third phase is left for CPU implementation as it is sequential in nature. Fig. 1 illustrates the design using the same example as in [1] where S1 to S5 represent the vertices of the subject polygon, C1 to C4 represent the clipping polygon and I1 to I4 represent the intersection points.

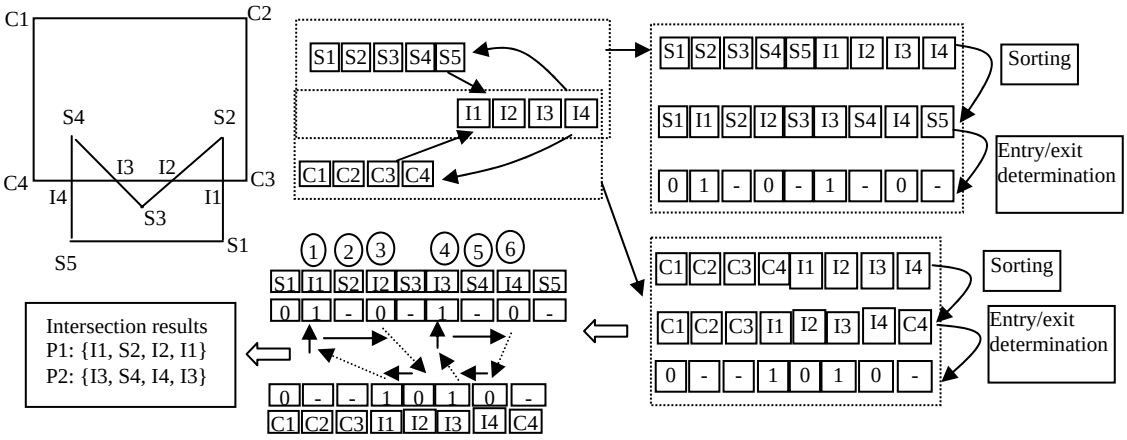

Fig. 1 Illustration of GPU-based Polygon Overlay Design in CudaGIS

For the GPU implementation of Phase 1, in a way similar to computing the intersection points between a polygon and scan lines [2], we assign a pair of subject and clipping polygon to a computing block. Each edge of a subject polygon is assigned to a thread and it will loop through the clipping polygon to compute the intersection points. The intersection points will be appended to both the vertex array of the subject polygon and the vertex array of the object polygon. These two arrays will be sorted based on a *v*

value which is the position index for a polygon vertex. For intersection point, *v* is calculated proportionally. If *ip* is the an intersection point between the two polygon vertices p_i and p_{i+1}, then *v* is calculated as i+distance(p,p_i)/distance(p,p_{i+1}). The sorting process can also be done in parallel on GPUs efficiently, especially within a computing block and shared memory is utilized.

Phase 2 marks each intersection point in the two arrays as either entry (1) or exit (0). We break down the phase into two steps so that each step can be realized in parallel. The first step is to determine whether the first vertex is inside or outside the subject/clipping polygon using a well known ray-casting algorithm [3] that can be easily parallelized on GPUs. The second step is to determine the entry/exit status for intersection points based on the inside/outside flag of the first polygon vertex by toggling the status of a previous intersection point. While this step seems to be serial in nature, by performing a prefix sum on the intersection points, the status can be easily computed based on the prefix sum value for all intersection points in parallel.

While the design has been implemented internally, the implementation has not been included in our CudaGIS [5] for two reasons. First, the current design and implementation have not considered special cases although they do not happen frequently in real world applications. Second, the design and implementation need to be extended to support other types of polygon overlay operations other than intersection. We note that CudaGIS is not designed to overlay two polygons. Instead, it is designed to overlay two large-scale polygon datasets. Quite a few excellent CPU based computational geometry packages (e.g., [5]) are available to compute the overlays of polygons. The CPU implementations probably are fast enough to overlay two reasonably large polygons in real world applications. Some indexing and preprocessing mechanisms are also available to speed up overlays of large polygons on CPUs. From a data management perspective, overlying two large-scale polygon datasets can be considered as a special type of spatial join in a way similar to nearest neighbor based or point-in-polygon test based spatial joins. Instead of returning pairs of identifiers, polygon overlay returns a set of newly computed polygons. Since the last step of overlaying two polygons already is done on CPUs, the variable length of output polygons should not be a problem. However, the newly generated polygons should be arranged using the simple array data structures discussed in [4] so that they can be streamed to GPU memory for next rounds of computations.

Reference:

1. G. Greiner and K. Hormann (1998). Efficient clipping of arbitrary polygons. ACM Transactions on Graphics, TOG 17(2):71-83.
2. Zhang J. (2011). Speeding Up Large-Scale Geospatial Polygon Rasterization on GPGPUs. Proceedings of ACM HPDGIS Workshop
3. Wikipedia. Point-in-Polygon. http://en.wikipedia.org/wiki/Point_in_polygon
4. Wikipedia. General Polygon Clipper (GPC). http://en.wikipedia.org/wiki/GPC_General_Polygon_Clipper_Library
5. J. Zhang and S. You (2012). CudaGIS: Report on the Design and Realization of a Massive Data Parallel GIS on GPUs. To appear in ACM SIGSPATIAL IWGS Workshop.