Data Structures for Parallel Spatial Algorithms on Large Datasets

(Vision paper)

W. Randolph Franklin
ECSE Dept, Rensselaer Polytechnic Institute
Troy, NY USA
mail@wrfranklin.org

Salles Viana Gomes de Magalhães
Universidade Federal de Viçosa
Viçosa – MG, Brasil
sallesviana@gmail.com

Marcus Vinícius Alvim Andrade
Universidade Federal de Viçosa
Viçosa – MG, Brasil
marcus.ufv@gmail.com

ABSTRACT

This paper describes data structures and algorithms for efficient implementation of GIS operations for large datasets on multicore Intel CPUs and on NVIDA GPUs. Typical operations are boolean combinations of polygons and map overlay. Efficient parallelization prefers simple regular data structures, such as structures of arrays of plain old datatypes. Warps of 32 threads are required to execute the same instruction (or be idle). Ideally, the data used by adjacent threads is adjacent in memory. Minimizing storage is important, as is accessing it in a regular pattern. That disparages pointers, linked lists, and trees. That implies that explicitly representing global topology is bad. If using only local topological formulae is sufficient, then it will be much faster. E.g., for many operations on a 2-D map (aka planar graph), the set of oriented edges suffices. Each edge knows the locations of its endpoints and the ids of its adjacent polygons. Any mass operation, such as area computation or point location, can be implemented as a map-reduce. All these techniques also apply in 3D to CAD/CAM and additive manufacturing. Indeed they are more important there.

CCS CONCEPTS

• **Theory of computation → Computational geometry**; Nearest neighbor algorithms; • **Computing methodologies → Parallel algorithms**; **MapReduce algorithms**;

KEYWORDS

parallel computation, computational geometry, map overlay, local topological formulae

ACM Reference Format:
W. Randolph Franklin, Salles Viana Gomes de Magalhães, and Marcus Vinícius Alvim Andrade. 2018. Data Structures for Parallel Spatial Algorithms on Large Datasets: (Vision paper). In *7th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data (BigSpatial 2018), November 6, 2018, Seattle, WA, USA*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3282834.3282839

BigSpatial 2018, November 6, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-6041-8/18/11…$15.00
https://doi.org/10.1145/3282834.3282839

1 PARALLEL COMPUTATION

The complex structure of sequential geometry algorithms causes the design of parallel geometry algorithms to be challenging. Nevertheless, it is apposite because all modern computers, even the Raspberry Pi and Apple iPhone, are multicore, and perhaps one third of all PCs have Nvidia GPUs, which are parallel computers. The Message Passing Interface (MPI) is popular. However this paper argues that MPI and distributed computing are often not necessary; that many problems considered to be big data can be processed on inexpensive lab servers.

We will consider two major parallel computing hardware architectures. The first is the multicore Intel Xeon CPU with large main memory, such as a quad 18-core with 144 hyperthreads and 2TB of main memory. Consider how much data fits into 2TB of DRAM. The memory is flatly addressable by any of the cores, with due consideration for caching and thread affinity. That amount of available shared memory suffices to hold many problems often considered to need external storage. A typical API for programming in this model is OpenMP. The cores asynchronously execute separate instruction streams. The parallel algorithm design requirement is to be decomposable into many separate threads that minimally write to the same memory addresses.

The second major parallel computing paradigm uses NVIDIA Tesla GPU accelerators with 5000 cores and 32GB of memory, such as the current Volta architecture. Efficient GPU programming requires the algorithm to utilize thousands of lightweight threads. Every thread in a warp of 32 threads must execute the same instruction, or be idle. A serial code fragment will use only 3% of the cores in its warp; Amdahl's law controls. Ideally, consecutive threads access consecutive words in memory.

Many current geometric data structures and paradigms are not amenable to parallelization. Examples include sweep lines, and most of the Computational Geometry Algorithms Library (CGAL) [9]. The first several levels of divide-and-conquer are not very parallelizable, as are hierarchical data structures like quadtrees and R-trees. Such data structures are to be deprecated in favor of flatter more regular ones like *Uniform Grids* and the *Corner Table* representation of triangle and tetrahedron meshes [21, 41].

Expressing algorithms in a map-reduce model can make them clearer, more concise, and more parallelizable. Reduction has been a component of high level languages at least since the APL language, proposed in 1957 and implemented by IBM in 1965. An efficient C++ map-reduce implementation targeting NVIDIA GPUs is CUDA Thrust[35]. Its other possible backends include OpenMP, Intel Threaded Building Blocks (TBB), and generic CPUs. Thrust is at a sweet spot of reasonable efficiency combined with hiding low

level CUDA details. A recent more powerful tool is Kokkos [46]. Parallel programming at a lower level is analogous to writing in assembly instead of in C++.

The range of problems easily solvable with this paradigm, as shown in the example libraries, is surprising. They include bucket sort (basically uniform grid), run length encoding and decoding, lexicographic sorting, discrete voronoi, and welding triangle vertices together, to eliminate redundant vertex positions and shared edges to produce a connected mesh.

2 PRIOR ART

Current Computational Geometry algorithms, as embodied in, e.g., CGAL, often solve the edge intersection problem with data structures like topological sweep lines[43] with optimal worst case execution time under a pair-comparison cost model. More complex geometrical adjacencies and intersections use a hierarchical tree structure, either an octree or R-tree[7], but they do not parallelize easily or well, expecially on GPUs. Various point location algorithms are presented in [10, 36, 38]. One has $\theta(\lg N)^2$ query time, another $\theta(\lg N)$ query time but $\theta(N^2)$ preprocessing time and space. Parallelizing the plane sweep algorithm is discussed in [31]. An example of modeling one application with multiple overlaid meshes is a physical dissection anatomical simulator [37]. An application of Franklin's local topological formulae to collisionless plasma simulations is presented in [25]. Our ideas are used in [8, 39]. Cluster-based parallel map overlay is presented in [1]. Problems with designing geometric primitives in additive manufacturing, where a medium lattice structure might have billions of elements, are discussed in [4, 42]. External algorithms for datasets too to large process internally, including cache-oblivious and optimal I/O, are presented by [5, 6, 45].

Early studies of parallel computational geometry are [2, 22–24, 32–34, 40, 48]. Large geometric datasets for testing are available at [3, 26, 44].

3 PRELIMINARIES

Consider a polygon, P, in the 2D plane. Intuitively, it has vertices and edges. If we restrict the number of each to be finite, then there will be no problem with infinite sequences, limits and convergence. The mathematical subject of analysis (used by calculus) is irrelevant. However some interesting objects are now prohibited. E.g., we cannot represent a curved object as the limit of a sequence of ever more complex straight-edges objects.

An edge is a connected segment of a straight line. That is, it cannot have two separate parts. Do we want to allow it to be infinite? There are arguments both ways. All Voronoi diagrams have some infinite edges. However, this makes representing edge lengths and polygon areas trickier, since they can be infinite also.

An edge has a vertex at each end, aka each edge is adjacent to two vertices. If we choose to prohibit isolated vertices, then a vertex is adjacent to one or more edges.

Two edges with a common vertex intersect at that vertex. Do we want to allow two edges to intersect elsewhere? Usually not, but allowing that permits star polygons and four new regular, stellated, polyhedra.

The vertices and edges form the *boundary* of P, ∂P. (The most accessible source for the mathematical terms in this paper is Mathworld [47]). P also has an interior, but on which side of ∂P? The conventional answer is the small finite part, not the large infinite part, but sometimes we want the other one.

Next, what operations do we want to perform on legal polygons? The obvious operations on polygons P and Q are *union*, $(P + Q)$, *intersection*, $(P \cdot Q)$, and *exclusive-or*, $(P \oplus Q)$. The *difference* operation, *A-and-not-B*, $(P - Q)$, is a common extra operation. Electrical engineers like the fact that all the other operations can be defined in terms of the difference, and so fewer types of electrical components are needed in circuit design. Allowing the unary operation of *complement*, P' or $\bar{P}$, opens up some possibilities. It is theoretically desirable to require that the set of all possible polygons be *closed* under these operations. That means that whenever these operations are applied to one or two legal polygons, then the result is always a legal polygon. This implies that: (1) The *empty set*, $\emptyset$, and its complement, the *universal set*, must be legal polygons. Indeed, $\emptyset$ results from intersecting two disjoint polygons. (2) A legal polygon may have multiple separate disjoint components, which may even be nested, like an island in a lake on the mainland.

A broader advantage of having the set of legal polygons closed under union or intersection, and the empty and universal sets being legal polygons, (and since those operations are both associative), is that the set of polygons, together with either operation, forms a mathematical *commutative group* in abstract algebra. Now, all the many theorems that have been proved about groups also apply to polygons under intersection or union.

4 GLOBAL TOPOLOGY OF A POLYGON

When a polygon's edges form more than one *connected component*, then each component of the edges and vertices is called a *loop*. The loops' containment relations form a forest of trees. E.g., consider the polygon that is the land area of Canada. It has a loop for the mainland shoreline, a second loop for Baffin Island's shore, a third loop for Lake Erie's shore, a fourth loop for Pelee Island in Lake Erie, etc. The fourth loop is inside the third loop, which is inside the first. The second loop is inside none of them, nor are any of them inside it. Conventionally, we would represent these containment relations explicitly.

We argue that that is unnecessary, apart from a few special cases. They include visually rendering a polygon on a graphics device, i.e., shading each region, and interpolating from loops of elevation contours to an elevation grid, [19, 20].

5 LOCAL TOPOLOGY OF A POLYGON

There are several advantages to minimizing the number of types of information to be stored about a geometric object. In the common case where I/O time dominates, even small constant factors in the space are important. There may be hard limits in the available storage, e.g., the biggest current GPU accelerator, the Nvidia Tesla V100, has 32GB of storage. Spilling over that limit would require data to be paged from the CPU memory, at a serious performance impact. Minimizing the number of types often simplifies the algorithm. When many data types are stored, they are often redundant; so consistency relations must be maintained as the data is manipulated. All this is more important in 3D.

What relations need to be stored explicitly depends on the intended operation. For the above Boolean operations, viz, union, intersection, difference, and complement, a mere *set of oriented edges* suffices. Start by observing that this representation suffices to determine whether a point p is contained in the polygon P. Extend semi-infinite ray up from p and count intersections. p is inside P iff that number is odd. Simulation of Simplicity (SoS) [12] handles geometric degeneracies, such as when the ray runs through a vertex. PinMesh [29] implements this in 3D (which is much harder than 2D). Preprocessing a sample dataset with 50 million triangles took only 14 elapsed seconds on a 16-core Xeon processor. The mean query time was 0.6 microseconds.

To compute $P + Q$, observe that *(1)* The output vertices are a subset of the input vertices and all the intersections of input edges. Those input vertices that are output vertices are exactly those that are outside the other polygon. *(2)* To find the output edges, cut the input edges where they intersect each other. Select those pieces of each polygon that are outside the other polygon.
Both the input and output are a set of edges, together with the vertices at their ends. No more global topology is needed.

Indeed, not even the complete edges are always needed. Suppose that our desired operation is to compute properties of the polygon such as area and edge length. Then, representing the polygon as a set of the *vertex-adjacencies* suffices. For each such adjacency, store this triple: the vertex's position, the direction that the edge leaves the vertex, and which side of the edge is the inside of the polygon. Each vertex and each edge induces two such triples, but they are not explicitly associated in this representation. This representation is so local that it does not even have complete vertices or edges, although that information could be computed. More details, including the formulae, are in [14, 34].

To intersect two polygons represented by sets of their edges, computing the above vertex-edge incidence representation is easier than computing the output edges, since we do not need to sort the intersections along each edge to cut it into segments. This provides an efficiency if we want the area of the union or intersection, but not the union or intersection itself.

6 FASTER COMPLEX ALGORITHMS

With the simple data structures, we can solve complex geometric problems with much more efficient algorithms than would otherwise be possible. Here are some examples.

Union of many polygons: This problem is to find the union of N polygons; perhaps $N = 100M$. The usual algorithm to unite N polygons is to start by uniting pairs, then to unite the pair polygons, and so on, building up a computation tree of height $\Theta(\lg N)$. With our local representation, the output can be computed as follows: *(1)* Find all the intersections of any two input edges. *(2)* Test all those points, together with all the input vertices, to see which are outside all the input polygons. *(3)* Each point that passes is an output vertex. Determine the two resulting vertex-edge incidences. Depending on the desired output data structure, combine them in pairs to form output edges.
For a wide distribution of inputs, the expected time is *linear* in the size of the input plus the output.

Area of the union of many polygons: One application of this is to compute the amount of material removed by a milling tool, where its path is represented by a set of overlapping polygons. The usual algorithm goes as follows: *(1)* Compute the union polygon. *(2)* Find its area.

Our algorithm is a simplified version of our explicit union algorithm in the previous section. The difference is that, instead of pairing up vertex-edge incidences to form edges, apply our polygon area formula to the vertex-edge incidences. I.e., finding the area of the union is now faster and simpler than finding the explicit union. An analysis of the 3D case is [14] and a parallel implementation for identical cubes, for a multicore CPU is [15].

Planar Graphs (Maps): The above ideas are even more useful when operating on embedded planar graphs, aka maps or meshes. The notation is not standardized, so we will use the terms *vertices*, also known as nodes and points, *edges*, and *polygons* aka faces. Typical operations include determining which polygon contains a test point, and intersecting, aka overlaying, two maps to produce a new map.

The easy way to locate which polygon contains point p is to test p against every polygon in turn. However, with this representation, we can run a ray up from p until it hits the first edge, and from that know which polygon contains p. The expected time is *constant* per query point, after expected linear processing time. We have implemented this in parallel in 3D [29].

Overlaying Maps: Overlaying two maps is also easy with this representation; see [27, 28]. The expected time, as usual, is linear in the input maps' sizes.

Overlaying 3D triangulations: This is conceptually similar, just harder, especially the implementation, [11, 30], which can process tens of millions of triangles much faster than other implementations.

Cross Areas: This is the problem of computing the areas of all the nonempty intersections when two maps are overlaid. One application is to interpolate a property of the polygons of one map over to the polygons of the other map. One example is to go from the known population of each census polygon of one map to estimate the population of each hydrography polygon of the other map.

This is a beautiful example of the power of the local topological formulae. The area of an output polygon requires only the set of its vertices together with the directions of the two edges intersecting there and which input polygon is on each side of each of those edges. Each intersection of an input edge of one map with an edge of the other creates a vertex common to four output polygons. Their areas can be accumulated in a hash table as the vertices are computed. The algorithm and implementation are presented in [13, 17, 18].

A 3D algorithm for the volumes of intersecting regions of two tetrahedrulations is [16].

7 SUMMARY AND FUTURE PLANS

Less is better (up to a point) has many advantages when representing GIS geometry. We must fight the urge to continually discover new topological relations. We must fight the urge to unnecessarily complexify simple data structures. The impact extends from 2D and GIS to 3D and Computer Aided Design. We are currently demonstrating that with new implementations in those domains, including more on GPUs, and on larger datasets.

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1117277.

REFERENCES

[1] Dinesh Agarwal, Satish Puri, Xi He, and Sushil K Prasad. 2012. A system for GIS polygonal overlay computation on linux cluster-an experience and performance report. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 1433–1439.

[2] Alok Aggarwal, Bernard Chazelle, Leo Guibas, Colm O'Dunlaing, and Chee Yap. 1985. Parallel Computational Geometry. In *Foundations of Computer Science – 25th Annual Symposium*. 468–477.

[3] AIM@SHAPE-VISIONAIR Shape Repository 2016. AIM@SHAPE-VISIONAIR Shape Repository. Retrieved 2016-02-02 from http://visionair.ge.imati.cnr.it//

[4] George Allen. 2014. CAD Implications of Additive Manufacturing (Presentation). In *GDM 2014 Workshop: Geometric Design Facing Manufacturing*. http://www.cs.technion.ac.il/gdm2014/Presentations/GDM2014_allen.pdf (retrieved 2016-11-15).

[5] Lars Arge and Mikkel Thorup. 2015. RAM-Efficient External Memory Sorting. *Algorithmica* 73, 4 (2015), 623–636. https://doi.org/10.1007/s00453-015-0032-8

[6] Lars Arge, D. E. Vengroff, and J. S. Vitter. 1995. External-memory algorithms for processing line segments in geographic information systems. In *Proc. Annual European Symposium on Algorithms, LNCS 979*. 295–310.

[7] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. 1998. An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. *J. ACM* 45, 6 (Nov. 1998), 891–923.

[8] Samuel Audet, Cecilia Albertsson, Masana Murase, and Akihiro Asahara. 2013. Robust and Efficient Polygon Overlay on Parallel Stream Processors. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'13)*. ACM, New York, NY, USA, 304–313. https://doi.org/10.1145/2525314.2525352

[9] CGAL. 2018. Computational Geometry Algorithms Library. Retrieved 2018-09-09 from https://www.cgal.org

[10] Mark de Berg, Otfried Cheong, and Marc van Kreveld. 2008. *Computational Geometry: Algorithms and Applications* (3 ed.). Springer.

[11] Salles Viana Gomes de Magalhães. 2017. *Exact and parallel intersection of 3D triangular meshes*. Ph.D. Dissertation. Rensselaer Polytechnic Institute.

[12] Herbert Edelsbrunner and Ernst Peter Mücke. 1990. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM T. Graphics* 9, 1 (January 1990), 66–104.

[13] Wm Randolph Franklin. 1990. Calculating Map Overlay Polygon' Areas Without Explicitly Calculating the Polygons — Implementation. In *4th International Symposium on Spatial Data Handling*. Zürich, 151–160.

[14] W. Randolph Franklin. 2004. Analysis of Mass Properties of the Union of Millions of Polyhedra. In *Geometric Modeling and Computing: Seattle 2003*, M. L. Lucian and M. Neamtu (Eds.). Nashboro Press, Brentwood TN, 189–202.

[15] Wm. Randolph Franklin. 2005. Mass Properties of the Union of Millions of Identical Cubes. In *Geometric and Algorithmic Aspects of Computer Aided Design and Manufacturing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Ravi Janardan, Debashish Dutta, and Michiel Smid (Eds.). Vol. 67. American Mathematical Society, 329–345.

[16] Wm Randolph Franklin and Mohan S. Kankanhalli. 1993. Volumes From Overlaying 3-D Triangulations in Parallel. In *Advances in Spatial Databases: Third Intl. Symp., SSD'93*, D. Abel and B.C. Ooi (Eds.). Lecture Notes in Computer Science, Vol. 692. Springer-Verlag, 477–489.

[17] Wm Randolph Franklin and Venkatesh Sivaswami. 1990. OVERPROP — Calculating Areas of Map Overlay Polygons without Calculating the Overlay. In *Second National Conference on Geographic Information Systems*. Ottawa, 1646–1654.

[18] Wm Randolph Franklin, Venkateshkumar Sivaswami, David Sun, Mohan Kankanhalli, and Chandrasekhar Narayanaswami. 1994. Calculating the Area of Overlaid Polygons Without Constructing the Overlay. *Cartography and Geographic Information Systems* (April 1994), 81–89.

[19] Michael Gousie and W. Randolph Franklin. 2003. Constructing a DEM from Grid-based Data by Computing Intermediate Contours. In *GIS 2003: Proceedings of the Eleventh ACM International Symposium on Advances in Geographic Information Systems* (November 7–8), Erik Hoel and Phillippe Rigaux (Eds.). New Orleans, 71–77.

[20] Michael B. Gousie and Wm. Randolph Franklin. 2005. Augmenting Grid-based Contours to Improve Thin Plate DEM Generation. *Photogrammetric Engineering & Remote Sensing* 71, 1 (2005), 69–79.

[21] Topraj Gurung and Jarek Rossignac. 2009. SOT: compact representation for tetrahedral meshes. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*. 79–88.

[22] Mei-Cheng Hu and James D. Foley. 1985. Parallel Processing Approaches to Hidden Surface Removal in Image Space. *Computers and Graphics* 9, 3 (1985), 303–317.

[23] Mohan Kankanhalli. 1990. *Techniques for Parallel Geometric Computations*. Ph.D. Dissertation. Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute.

[24] Mohan Kankanhalli and Wm Randolph Franklin. 1995. Area and Perimeter Computation of the Union of a Set of Iso-Rectangles in Parallel. *J. Parallel Distrib. Comput.* 27, 2 (June 1995), 107–117. https://doi.org/10.1006/jpdc.1995.1076

[25] Julian Kates-Harbeck, Samuel Totorica, Jonathan Zrake, and Tom Abel. 2015. Simplex-in-Cell Technique for Collisionless Plasma Simulations. Retrieved 2016-11-15 from https://arxiv.org/pdf/1506.07207.pdf

[26] Large Geometric Model Archive 2016. GIT Large Geometric Model Archive. Retrieved 2016-02-02 from http://www.cc.gatech.edu/projects/large_models/

[27] Salles Viana Gomes Magalhães. 2015. An Efficient Algorithm for Computing the Exact Overlay of Triangulations. In *Proc. 2nd ACM SIGSPATIAL PhD Workshop (SIGSPATIAL PhD '15)*. ACM, New York, NY, USA, Article 3, 4 pages. https://doi.org/10.1145/2855680.2855840

[28] Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, and Wenli Li. 2015. Fast exact parallel map overlay using a two-level uniform grid. In *4th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data (BigSpatial)*. Bellevue WA USA. https://doi.org/10.1145/2835185.2835188

[29] Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, and Wenli Li. 2016. PinMesh – Fast and exact 3D point location queries using a uniform grid. *Computer & Graphics Journal, special issue on Shape Modeling International 2016* 58 (Aug. 2016), 1–11. https://doi.org/10.1016/j.cag.2016.05.017 (online 17 May). Awarded a reproducibility stamp, http://www.reproducibilitystamp.com/.

[30] Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin, Wenli Li, and Maurício Gouvêa Gruppi. 2016. Exact intersection of 3D geometric models. In *Geoinfo 2016, XVII Brazilian Symposium on GeoInformatics*. Campos do Jordão, SP, Brazil.

[31] Mark McKenney, Roger Frye, Mathew Dellamano, Kevin Anderson, and Jeremy Harris. 2016. Multi-core parallelism for plane sweep algorithms as a foundation for GIS operations. *GeoInformatica* (2016), 1–24. https://doi.org/10.1007/s10707-016-0277-7

[32] Chandrasekhar Narayanaswami. 1991. *Parallel Processing for Geometric Applications*. Ph.D. Dissertation. Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute. UMI no. 92-02201.

[33] C. Narayanaswami and Wm Randolph Franklin. 1991. Determination of mass properties of polygonal CSG objects in parallel. *Internat. J. Comput. Geom. Appl.* 1, 4 (1991), 381–403.

[34] Chandrasekhar Narayanaswami and Wm Randolph Franklin. 1991. Determination of Mass Properties of Polygonal CSG Objects in Parallel. In *Proc. Symposium on Solid Modeling Foundations and CAD/CAM Applications*, Joshua Turner (Ed.). ACM/SIGGRAPH, 279–288.

[35] Nvidia. 2015. CUDA Toolkit Documentation. Retrieved 2016-02-23 from http://docs.nvidia.com/cuda/thrust/

[36] Joseph O'Rourke. 1998. *Computational Geometry in C* (2 ed.). Cambridge University Press.

[37] Junjun Pan, Junxuan Bai, Xin Zhao, Aimin Hao, and Hong Qin. 2014. Dissection of Hybrid Soft Tissue Models Using Position-based Dynamics. In *Proceedings of the 20th ACM Symposium on Virtual Reality Software and Technology (VRST '14)*. ACM, New York, NY, USA, 219–220. https://doi.org/10.1145/2671015.2671129

[38] F. P. Preparata and M. I. Shamos. 1985. *Computational Geometry: An Introduction*. Texts and Monographs in Computer Science Springer-Verlag.

[39] Satish Puri and Sushil K. Prasad. 2013. Efficient Parallel and Distributed Algorithms for GIS Polygonal Overlay Processing. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW '13)*. IEEE Computer Society, Washington, DC, USA, 2238–2241. https://doi.org/10.1109/IPDPSW.2013.174

[40] John H. Reif and Sandeep Sen. 1988. An Efficient Output-sensitive Hidden-Surface Removal Algorithm and its Parallelization. In *Proc. Fourth Annual Symposium on Computational Geometry*. 193–200.

[41] Jarek Rossignac, Alla Safonova, and Andrzej Szymczak. 2001. 3D Compression Made Simple: Edgebreaker on a Corner-Table. In *Shape Modeling International*. 278–283. https://doi.org/10.1109/SMA.2001.923399

[42] Vadim Shapiro. 2014. Geometric Modeling of Material (Micro)Structures (Presentation). In *GDM 2014 Workshop: Geometric Design Facing Manufacturing*. http://www.cs.technion.ac.il/gdm2014/Presentations/GDM2014_shapiro.pdf (retrieved 2016-11-15).

[43] Diane Souvaine. 2008. Line Segment Intersection Using a Sweep Line Algorithm. Retrieved 2016-11-13 from http://www.cs.tufts.edu/comp/163/notes05/seg_intersection_handout.pdf

[44] Stanford Scanning Repository 2016. The Stanford 3D Scanning Repository. Retrieved 2016-02-02 from http://graphics.stanford.edu/data/3Dscanrep/

[45] Laura Toma, Lars Arge, and J. S. Vitter. 2001. I/O–efficient algorithms for problems on grid-based terrains. *ACM J. Experimental Algorithmics* 6 (2001).

[46] Christian Trott and Glen Hansen. 2018. Kokkos C++ Performance Portability Programming EcoSystem. Retrieved 2018-10-05 from https://github.com/kokkos

[47] Eric W. Weisstein. 2016. MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/.

[48] Chee-Keng Yap. 1987. What can be Parallelized in Computational Geometry?. In *International Workshop on Parallel Algorithms and Architectures*.