Illustrating the Streaming Construction of 2D Delaunay Triangulations

Martin Isenburg
Comp. Sci. Division
UC Berkeley, CA, USA

Yuanxin (Leo) Liu
Dept. Comp. Sci.
UNC Chapel Hill, NC, USA

Jonathan Shewchuk
Comp. Sci. Division
Berkeley, CA, USA

Jack Snoeyink
Dept. Comp. Sci.
UNC Chapel Hill, NC, USA

Categories and Subject Descriptors: I.3.5 Computational Geometry and Object Modeling: Geometric algorithms, languages, and systems

General Terms: Algorithms

Keywords: streaming computation, out-of-core computation, Delaunay triangulation, Voronoi diagram, quadtree, memory management.

1. INTRODUCTION

Advances in sensor technology support the collection of data sets far larger than the memory capacity of a personal computer. For example, the North Carolina Floodplain Mapping project[1] has used LIDAR (airborne laser) to capture elevation to assess flood risks, set insurance premiums, and create disaster plans for the entire state. The bare-earth data for the Neuse River Basin shown in the accompanying video consists of over 500 million points, totaling 12 GB.

Many approaches are taken to develop algorithms to process such large data sets. Practitioners may simply divide and conquer: partition the problem into smaller pieces, then find a way to merge the solutions. Theoreticians develop I/O-efficient algorithms [8, 1] to minimize the cost of reading from and writing to disk, or cache-efficient algorithms [6] to minimize the cost of memory accesses in a hierarchical memory model.

We advocate *streaming*: making a small number of sequential passes over a data file (ideally, one pass) and processing the data using a memory buffer whose size depends upon the amount of *spatial coherence* in the data and the algorithm. Thus, we use data formats that document spatial coherence, and algorithms that exploit it, described in the next section. With them, we compute a billion-triangle terrain representation for the Neuse River Basin from 12 GB of LIDAR data in an hour using less than 100 MB of memory on a laptop [5]. The output can stream to further processing, such as constructing contour or raster digital elevation maps.

2. STREAMING AND FINALIZATION

Streaming formats seek to present data in an order that supports processing as the data arrives. In familiar streaming formats for audio and video, decompression and filtering operations buffer data in a sliding window whose size is chosen so that each operation has all the local information that it needs. This allows data to stream out as fast as it streams in, with only a little latency.

For point clouds and triangle meshes, many operations depend only on data in a local neighborhood. Unfortunately, for most operations, there is no stream order for which a small sliding window always contains all the neighbor information needed. So we add explicit "finalization tags" to our input and output streams; these tags tell a streaming algorithm which geometric entities will not be addressed by the stream again. This peek into the future of a stream can help an algorithm to keep its memory footprint small.

Isenburg and Lindstrom [4] describe a streaming format for polygon meshes that is an intermixed stream of vertex coordinates, polygons that reference vertices that have already appeared in the stream, and *vertex finalization tags* that indicate when all the polygons referencing a vertex have already appeared. Finalization of vertex v tells the application that it can complete all computations that were waiting for v's topology, output partial results, and safely free any data structures that are no longer needed.

To support smoothing and normal estimation for streaming points scanned from 3D surfaces, Pajarola [7] suggests finalizing a point after its k nearest neighbors have arrived. He initially sorts the points into a spatial total order that allows him to determine when a point can be finalized.

To support Delaunay triangulation for streaming points without full sorting, we define *spatial finalization* for point streams [5]: we partition space into regions, and each region is finalized by a *spatial finalization tag* after the last point in the region appears in the stream. In this video, we focus on 2D point streams. The *width* of a stream is the maximum number of points that appear at any one time in regions that are not yet finalized. The width determines the maximum number of points that must be buffered simultaneously.

Ideally, spatial finalization tags are already present in the input to a process, but if not, they can be added with two read passes over the input points. The first pass simply determines the bounding box, which we partition into regions with a quadtree. The second pass counts how many points fall into each leaf quadrant. It can also sample a point from each quadrant to help construct a biased randomized insertion order (BRIO) [2]. A third pass, knowing the counts, can produce a stream with a finalization tag whenever the last point of a quadrant appears in the stream.

If the input is not spatially coherent, then we must use a spatial sort to increase coherence and decrease stream width. However, we have observed that huge real-world data sets already have a lot

[1] http://www.ncfloodmaps.com

Copyright is held by the author/owner.
SCG'06, June 5–7, 2006, Sedona, Arizona, USA.
ACM 1-59593-340-9/06/0006.

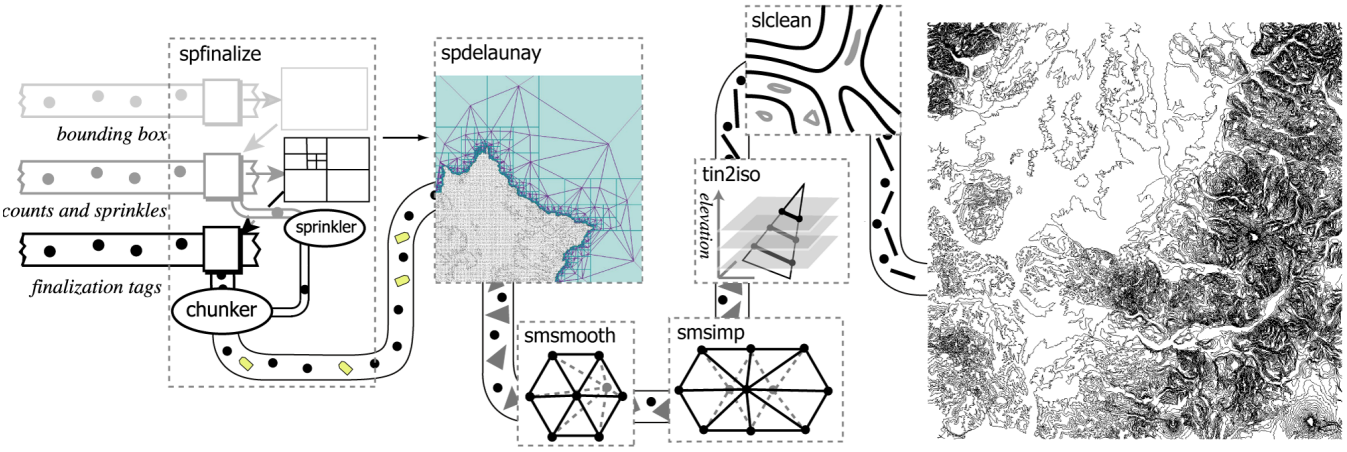

Figure 1: Spatially finalized points stream out of `spfinalize` into `spdelaunay`, which produces a Delaunay triangulation in a streaming mesh format [5]. This TIN, or triangulated terrain model, can be piped to further processing: `smsmooth` smoothes TIN z-coordinates, then `smsimp` simplifies the TIN to a smaller size, then `tin2iso` extracts specified isocontours as a streaming line format. `slclean` removes short polylines and streams the result to a viewer that can visualize output while input is still being read.

of spatial coherence. (This is not surprising, since otherwise the programs creating them would have been bogged down by thrashing.) We can improve coherence and decrease stream width by *chunking*—reordering buffered points so all the points in a quadrant appear consecutively. Then, within each quadrant we apply the biased random sampling (BRIO) of Amenta et al. [2] to avert the worst-case behavior of incremental Delaunay triangulation. These techniques allow us to run our algorithm without needing a full initial sort.

3. STREAMING TRIANGULATION

We modified an incremental Delaunay triangulator to use finalization tags. We maintain two data structures that can be seen in the accompanying video: the triangulation, and a dynamic quadtree that tracks which quadrants have been finalized, and makes circumcircles wait on unfinalized quadrants that they intersect. When `spdelaunay2d` reads a point p, it inserts p into the Delaunay triangulation. When it reads a finalization tag, it notes the finalized cell in the quadtree, and for any circumcircles for which it cannot find an intersecting unfinalized cell, it writes the associated triangles to the output stream, and frees memory.

Our pipeline, illustrated on the left side of Figure 1, uses two simultaneously running processes to triangulate the Neuse River Basin in 61 minutes and 100 MB of memory. The *finalizer* reads a 12 GB stream of raw points three times from disk; as mentioned above, the first two passes may be omitted if the bounding box and grid cell counts are already available. During the third pass it inserts finalization tags, chunks the points to improve width, samples with a BRIO, and streams its output to the triangulator, which writes a 24 GB mesh to the disk. No intermediate information is stored on disk. The finalizer occupies 50–60 MB of memory (used mainly to buffer points while reordering), while the triangulator occupies 23 MB—less than 0.1% of the size of the mesh. If the triangulator can read an already-finalized point stream from disk, then it uses only the 23 MB and runs in 49 minutes.

Since the output is a streaming triangulation, it may be piped to other applications, such as the smoothing, simplification, contour extraction, and visualization depicted in Figure 1. Because each of the processes after the finalizer makes a single streaming pass over the data, the isocontours begin to appear while the finalizer is still reading input. The accompanying video is captured from a demo is available at `http://www.cs.unc.edu/~isenburg/sd`.

Acknowledgments

This work was partially supported by NSF grant CCF-0429901 and NGA/DARPA HM1582-05-2-0003. Special thanks to Kevin Yi for providing us overnight FTP access to the 12 GB Neuse River Basin data. Their STREAM project uses I/O efficient algorithms to process large terrains [1].

4. REFERENCES

[1] P. K. Agarwal, L. Arge, and K. Yi. I/O-efficient construction of constrained Delaunay triangulations. In *Proc 13th ESA, LNCS* 3669, pages 355–366. Springer Verlag, October 2005.

[2] Nina Amenta, Sunghee Choi, and Günter Rote. Incremental constructions con BRIO. In *Proc 19th ACM SCG*, pages 211–219, June 2003.

[3] M. Isenburg and S. Gumhold. Out-of-core compression for gigantic polygon meshes. *ACM Transactions on Graphics*, 22(3):935–942, July 2003.

[4] M. Isenburg and P. Lindstrom. Streaming meshes. In *Visualization '05 Proceedings*, pages 231–238, 2005.

[5] M. Isenburg, Y. Liu, J. Shewchuk, and J. Snoeyink. Streaming computation of Delaunay triangulations. `http://www.cs.unc.edu/~isenburg/sd`, 2006.

[6] Piyush Kumar. Cache oblivious algorithms. In *Algorithms for Memory Hierarchies*, Meyer, Sanders and Sibeyn, eds., *LNCS 2625*, pages 193–212. Springer-Verlag, 2003.

[7] R. Pajarola. Stream-processing points. In *Visualization '05 Proceedings*, pages 239–246, 2005.

[8] J. S. Vitter. External memory algorithms and data structures: Dealing with MASSIVE data. *ACM Computing Surveys*, 33(2):209–271, 2001.