Efficient Out-of-Core Build of a Topological Data Structure
226

Sara McMains
Carlo Séquin
University of California, Berkeley
sara@cs.berkeley.edu

Many solid modeling applications require information not only about the geometry of an object but also about its “topology,” the connectivity of its faces, edges, and vertices. Many interchange formats do not provide this information; therefore, the application must derive it as it builds its own topological data structure from an unorganized list of triangles. For very large datasets, the data structure itself can be bigger than core memory, and a naive algorithm for building it can become prohibitively slow due to memory thrashing. In this sketch, we describe a new out-of-core algorithm that can build our topological data structure, LEDS (the Loop Edge-Use Data Structure), which closely related to Weiler's radial edge data structure,1 from very large datasets.

To determine connectivity relationships, we use hash tables to match up all coincident vertex coordinates and edge-uses. One possible approach is to build these hash tables and construct and update the LEDS as the input is read. Unfortunately, for a large file, each new element read could be connected to elements that have already been written out to disk, and those elements will now need to be paged back in to be updated. Even if the input is extremely coherent, so that updates to the same element are closely spaced in time, we will still see thrashing when the hash tables become too large to co-exist in memory. The problem of random update accesses is also encountered in bulk loading object-oriented databases; Wiener's algorithm in this domain2 inspired our approach.

Our algorithm for non-memory-resident data avoids thrashing in two ways: by re-ordering and grouping random hash-table accesses, so that we need to build and access only one memory-sized partition of a single larger hash table at a time, and by using external merge-sorts to re-order the intermediate data generated using the hash tables. This allows us to write all of the information that needs to be recorded in each entity at creation time, and we never need to go back and modify entities that have already been written out to disk. Our only out-of-order accesses are within the in-memory hash table partitions and during the sorting stage.

For testing, we took a simple curved shape and varied the fineness of the triangulation to produce files of different sizes (Figure 1). To extract the effects of input coherency, we made two versions of each file: one with the triangles organized in consecutive triangle strips, and one with the same triangles in random order. Running on an SGI Indy with 32M of RAM, the naíve approach is efficient for building a small memory-resident LEDS, with identical performance on random and coherent input. For larger files, however, performance degrades rapidly due to thrashing, particularly for the random files. For small files that fit in memory, the out-of-core algorithm takes approximately 20% longer to build the LEDS than the naive algorithm, but the intelligent use of virtual memory more than makes up for this overhead on larger files (Figure 2). The out-of-core algorithm we present makes building a very large topological data structure feasible, regardless of the coherence of the input.

References

1. Weiler, K. (1988). The radial edge structure: A topological representation for non-manifold geometric boundary modeling. Geometric Modeling for CAD Applications, 3-36.
2. Wiener, J.L. & Naughton, J.F. (1995). OODB bulk loading revisited: The partitioned-list approach. Proc. of the 21st International Conference on Very Large Data Bases, 30-4.

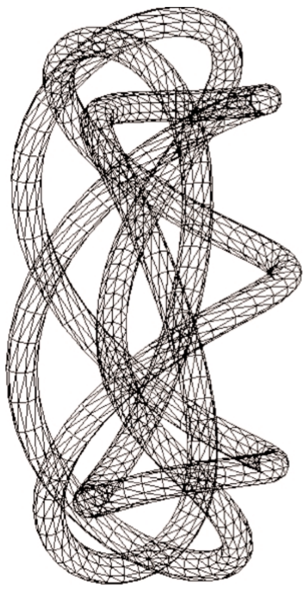
Figure 1: Our test part, coarsely tesselated to show triangle-strip organization.

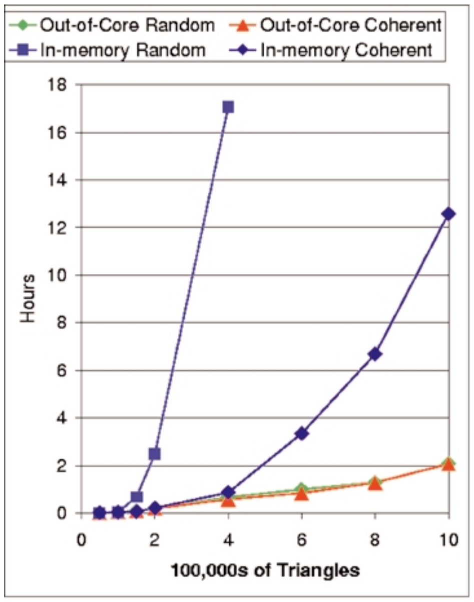

Figure 2: Naíve vs. out-of-core build times for the test part triangulated at various resolutions. Note that the results for the out-of-core algorithm on the coherent and randomly ordered versions of the input are coincident.