Experiments on Parallel Polygon Triangulation Using Ear Clipping

Günther Eder* Martin Held* Peter Palfrader*

Abstract

We present an experimental study of different strategies for triangulating polygons in parallel. As usual, we call three consecutive vertices of a polygon an ear if the triangle that is spanned by them is completely inside the polygon. Extensive tests on thousands of sample polygons indicate that about 50% of the vertices of most polygons form ears, which suggests that polygon-triangulation algorithms based on ear-clipping might be well-suited for parallelization. We discuss three different on-core approaches to parallelizing ear clipping and report on our experimental findings. Extensive tests show that the most promising method achieves an average speedup of about 3 on a quad-core processor.

1 Introduction

Three consecutive vertices (v_{i-1}, v_i, v_{i+1}) form an ear of a simple polygon P if $\overline{v_{i-1}, v_{i+1}}$ is a diagonal of P. Cutting along this diagonal removes the vertex v_i, the "base" of the ear, thus reducing the number of vertices of P by one. The basic idea of ear clipping is to iteratively cut off ears until the polygon has shrunk to a triangle. The algorithm's correctness hinges upon Meisters' two-ears theorem which states that every non-trivial simple polygon has at least two non-overlapping ears [3].

A typical ear-clipping algorithm operates in two phases. *Classification:* iterate along P to determine all instances of three consecutive vertices that form an ear of P. These potential ears are stored in a queue. *Clipping:* iteratively pick a candidate ear from the queue and clip it if it is still valid. As an ear (v_i, v_j, v_k) is clipped and stored in a triangle list, its two outer vertices v_i and v_k have to be checked to determine whether they form the bases of new ears now. Every new ear is added to the queue. The process ends for an n-vertex polygon when $n-3$ ears have been clipped.

Held's fast industrial-strength triangulation tool FIST [2] is a polygon triangulation framework based on ear-clipping. While the basic ear-clipping algorithm has an $\mathcal{O}(n^2)$ worst-case complexity, FIST employs multi-level geometric hashing to speed up the computation to near-linear time for almost all inputs.

*Universität Salzburg, FB Computerwissenschaften, 5020 Salzburg, Austria; supported by Austrian Science Fund (FWF) Grant P25816-N15; {geder,held,palfrader}@cosy.sbg.ac.at.

Surprisingly little work has been done on computing triangulations of simple polygons in parallel. The literature focuses mostly on (Delaunay) triangulations of point sets rather than polygons, see for instance Rong et al. [5] and Xin et al. [7]. In 2013, Qi et al. [4] introduced a primarily GPU-based algorithm to compute constrained Delaunay triangulations.

We analyzed the prevalence of ears in our vast set of test data and find that in most polygons about half the vertices form the bases of ears. If we look only at convex vertices then a vast majority (98 %) of them belong to ears. This suggests that clipping many ears simultaneously is feasible. Our contribution are three algorithms for parallelizing ear-clipping on a multi-core processor, which we implemented and tested extensively.

2 Parallel Ear-Clipping Algorithms

We discuss three algorithms for extending the classic FIST tool such that it can operate in parallel. The algorithms differ in how they split the polygon and its ears such that the subsequent ear clipping can be carried out in parallel.

Divide and Conquer. The basic idea is to split the polygon P into as many independent, equally sized sub-polygons as CPU cores are available. Since it seems costly to determine suitable diagonals that achieve balanced splits, we simply use vertical lines to split the polygon. Using a variant of the Sutherland-Hodgman polygon clipping algorithm [6], we can split a polygon along a line ℓ (which does not pass through a vertex of P) in time $\mathcal{O}(n)$, at a cost of at most $\mathcal{O}(n)$ Steiner points caused by the number of intersections between ℓ and the polygon boundary. In our tests, the number of Steiner points seems to be bounded by $\sqrt{n}$ for almost all but contrived inputs. We then run one (sequential) FIST instance per core to obtain a triangulation of each sub-polygon. Glueing the triangulations of all sub-polygons together yields a triangulation of P, albeit with Steiner points which have to be removed.

So consider a pair of Steiner points s_a, s_b that are consecutive along ℓ: We create a hole H by removing all triangles incident in s_a, s_b. Since s_a, s_b lie in the interiors of edges of the boundary of H and since H is "double-star-shaped", the hole H can be triangulated easily without re-creating triangles incident to s_a or s_b.

Partition and Cut. In this approach, we pick k equally-spaced vertices along the boundary of the polygon and partition the boundary into k chains, one for each thread. (Two adjacent chains then always share one of these selected vertices.) In order to run parallel classification steps, one for each chain, we use a per-thread queue to store potential ears instead of a single global queue. Then, in the clipping phase, each thread processes all ears from its queue. As usual, clipping the ear (v_{i-1}, v_i, v_{i+1}) involves checking whether $v_{i\pm1}$ has become the basis of a new ear. However it is only added to the queue if it is not a vertex shared with a neighboring chain. Since each thread only clips ears in its own chain and care is taken to never remove vertices shared between chains, both the classification and clipping step of each thread can run independently of other threads.

After all k queues are empty and the parallel clipping threads have ended, some part of the original polygon remains untriangulated. This part is then processed using a final, sequential run of FIST which completes the triangulation of the polygon.

Mark and Cut. The key observation of this approach is the following: Every second ear along the polygon's boundary is non-overlapping. Therefore, we can process every second ear and clip it independently from all other ears. The phases used by this approach thus differ slightly from the previous two algorithm: In the *mark phase*, we walk along the polygon boundary and store the index of every second vertex in an array A. In the *cut phase*, which can be run by many threads in parallel, we consider every vertex in A, and if it forms the basis of an ear we clip it immediately.

One thread is tasked with running the mark phase. As soon as it has processed half of the polygon boundary, the remaining threads launch a cut phase on the indices stored in A so far while the first thread continues until it has processed the remainder of the boundary.

Once all threads are finished, the cutting threads are re-launched on the vertices that have since been added to A. The marking thread now revisits what remains after the parallel ear-clipping on the first half of the polygon boundary. This continues until only a small number of ears are found in a cut phase. (In our tests we switched to the sequential FIST once fewer than 20 new triangles were generated in one cutting phase.) We then use one sequential run of FIST on the remaining polygon to finish the triangulation.

3 Experimental Results

We implemented all three parallel variants of FIST as an on-core parallelization by the use of OpenMP/C++. Our test system runs CentOS 6.5 on an 2014 Intel Xeon E5-2667 v3 CPU at 3.20 GHz with 8 cores and 132 GB RAM.

Our implementations were tested on about 20 000 polygons with up to four million vertices per input, consisting of both real-world and synthetic data that exhibits various characteristics. The test data was collected over the past 30 years by Martin Held and includes proprietary CAD/CAM designs, sampled printed-circuit board layouts, geographic maps, sampled spline curves and font outlines, closed fractal and space filling curves, as well as star-shaped and various types of "random" polygons generated by RPG [1].

In our tests, we compare the runtime of our parallel algorithms to the runtime of the sequential FIST tool. With eight cores a speedup of about 3 is observed with both the mark-and-cut and the partition-and-cut variants; cf. Fig. 1. The speedup of the divide-and-conquer approach is slightly lower. Most desktop computers have four cores and in such a setting the mark-and-cut variant produces a speedup of about 2–3. Our implementations are not (yet) tuned and, likely, there is room for further improvement.

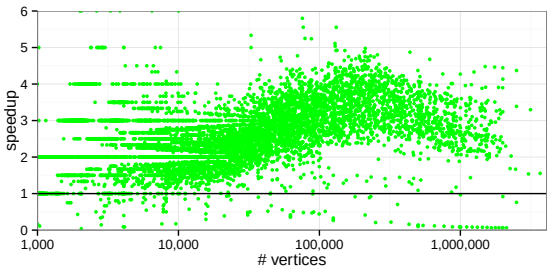

Figure 1: Speed-up obtained by Mark and Cut shown as a function of input size for eight cores.

References

[1] T. Auer and M. Held. Heuristics for the Generation of Random Polygons. In *Proc. 8th Canad. Conf. Comput. Geom (CCCG 1996)*, pages 38–44, Aug. 1996.

[2] M. Held. FIST: Fast Industrial-Strength Triangulation of Polygons. *Algorithmica*, 30(4):563–596, 2001.

[3] G. H. Meisters. Polygons have Ears. *The American Mathematical Monthly*, 82(6):648–651, June 1975.

[4] M. Qi, T. Cao, and T. Tan. Computing 2D Constrained Delaunay Triangulation Using the GPU. *IEEE Trans. Visualizat. Comput. Graph.*, 19(5):736–748, May 2013.

[5] G. Rong, T.-S. Tan, T.-T. Cao, and Stephanus. Computing Two-Dimensional Delaunay Triangulation using Graphics Hardware. In *Proc. ACM Symp. Interactive 3D Graphics*, I3D '08, pages 89–97, 2008.

[6] I. E. Sutherland and G. W. Hodgman. Reentrant Polygon Clipping. *C. ACM*, 17(1):32–42, Jan. 1974.

[7] S.-Q. Xin, X. Wang, J. Xia, W. Mueller-Wittig, G.-J. Wang, and Y. He. Parallel Computing 2D Voronoi Diagrams using Untransformed Sweepcircles. *Computer-Aided Design*, 45(2):483–493, 2013.