2021 KAIST Spring CS504 Term Project

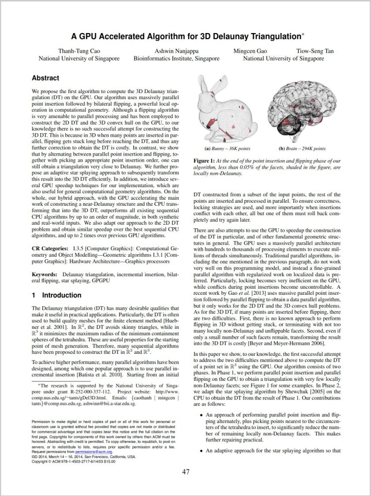
Date: 31 May 2021

gCrust: 2D curve reconstruction on the GPU

Team 3
Siwon Kim
Jaehwang Jung
Sungsoo Han

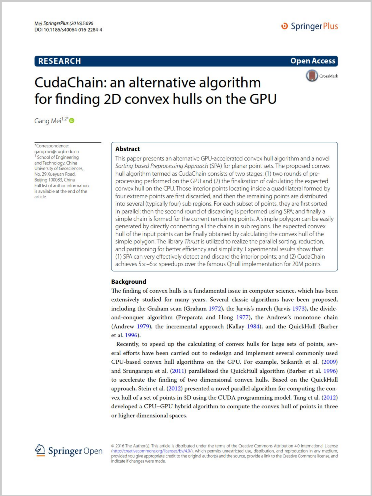
Background: Parallelization

Parallelizing algorithms on the GPU has been an important topic in Computational Geometry.



gDel3D (Cao et al., 2014) [1]
: Parallelizing 3D Delaunay Triangulation

gHull (Gao et al., 2013) [4]
: Parallelizing 3D Convex Hull



CudaChain (Gang Mei, 2016) [5]
: Parallelizing 2D Convex Hull

Background: 2D Curve reconstruction

Performance of 2D curve reconstruction is as important as its accuracy.

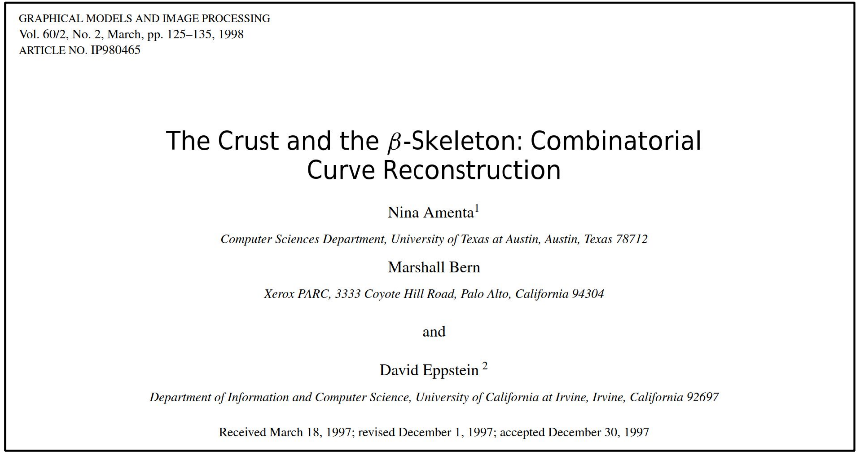
[Figure 1: Real-time 2D curve reconstruction helps to recognize broken down or defected traffic lanes]

Real-time surface reconstruction is used in variety of engineering applications, such as autonomous driving. In particular, the real-time 2D curve reconstruction can be used for traffic lane recognition.

Objective

We aim to parallelize the algorithm determining the *Crust* on the GPU.

The *Crust* is a widely used approach of a 2D curve reconstruction introduced in "The Crust and the Beta-Skeleton: Combinatorial Curve Reconstruction (Amenta et al., 1997) [2]".



GRAPHICAL MODELS AND IMAGE PROCESSING
Vol. 60/2, No. 2, March, pp. 125–135, 1998
ARTICLE NO. IP980465

The Crust and the β-Skeleton: Combinatorial Curve Reconstruction

Nina Amenta[1]

Computer Sciences Department, University of Texas at Austin, Austin, Texas 78712

Marshall Bern

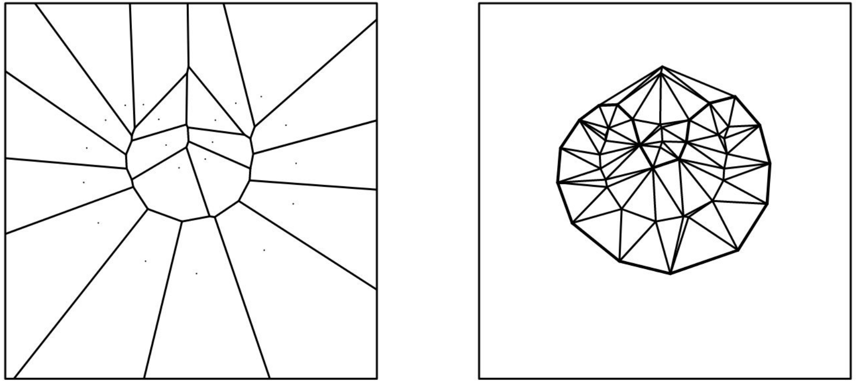
Xerox PARC, 3333 Coyote Hill Road, Palo Alto, California 94304

and

David Eppstein [2]

Department of Information and Computer Science, University of California at Irvine, Irvine, California 92697

Received March 18, 1997; revised December 1, 1997; accepted December 30, 1997



[Figure 2: Explanation of *Crust* [2]]

Algorithm Details: (1) Crust Revisited

> Definition of the *Crust*
>
> Let *S* be a finite set of points in the plane, and let *V* be the vertices of the Voronoi diagram of *S*. Let *S'* be the union *S* ∪ *V*, and consider the Delaunay triangulation of *S'*. An edge of the Delaunay triangulation of *S'* belongs to the *crust* of *S* if both of its endpoints belong to *S*.

As we can see from the definition, algorithm determining the *Crust* of a given point set is a composition of Delaunay Triangulation & Voronoi Diagram algorithms.

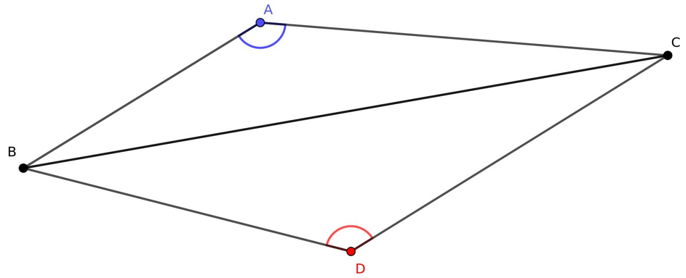
Originally, it uses the *Triangle* algorithm introduced by Shewchuk J.R. [3] for DT & VD construction, which is hard to parallelize on the GPU due to dependency between algorithmic steps.

In our work, we will use the *gDel2D* algorithm introduced by Cao et al. [1] for DT & VD construction, which outperforms 10 times better than the *Triangle*.

Algorithm Details: (2) gDel2D (Cao et al.)

gDel2D is a parallelized 2D Delaunay triangulation algorithm based on incremental insertion.

gDel2D parallelize each step of incremental insertion, (1) Point insertion & (2) Edge flipping, while reducing the total number of edge flipping.



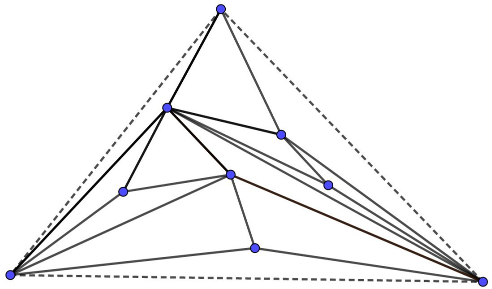
[Figure 3: Edge flip is required when *A* + *D* > *Pi*]

The main idea of reducing the total number of edge flipping is to minimize occurrence of *skinny* triangles.

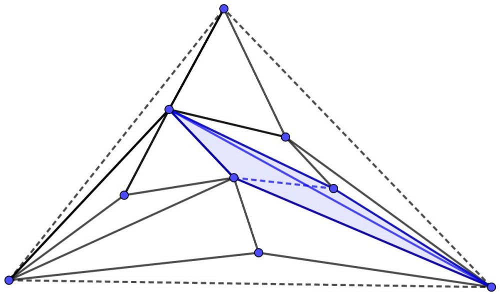
Algorithm Details: (2) gDel2D (Cao et al.)

There are two design choices: *InsAll* & *InsFlip*.

InsAll: Perform every insertion, then perform every edge-flip.
InsFlip: Perform insertion and edge-flip iteratively.



[Figure 4: After Point Insertion without any flipping]

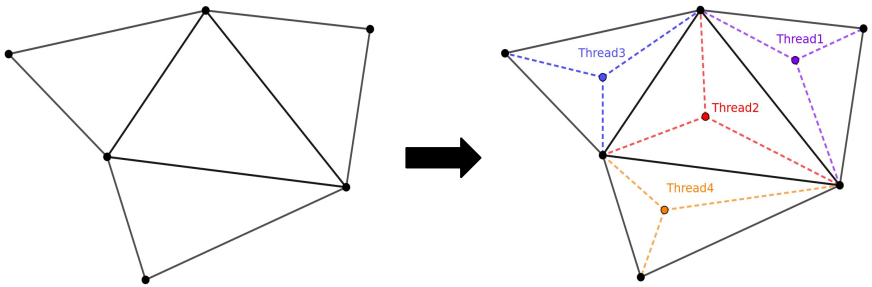


[Figure 5: Recursive edge-flipping may required]

InsAll may bring many skinny triangles after point insertion, and recursive edge-flipping may be required. Thus it is hard to parallelize edge-flip.

InsFlip requires more efforts on relocating points, but every skinny triangle is removed by edge-flipping after few iterations. Thus recursive edge-flipping is not required. This makes it easy to parallelize flipping and dominates inefficiency caused by relocation.

Algorithm Details: (2) gDel2D (Cao et al.)

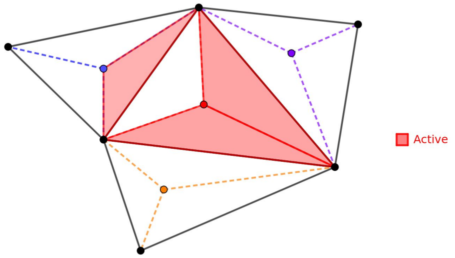


[Figure 6: Parallel Point Insertion]

Step 1) Parallelizing Point Insertion.

- Assigns a thread per each triangle.
- The key idea to reduce the number of edge flipping is inserting a point *nearest to the circumcenter* of the triangle by sorting.
  - Evenly distributes cases generating skinny triangles.
- Relocate & sort points according to newly generated triangles.

Algorithm Details: (2) gDel2D (Cao et al.)

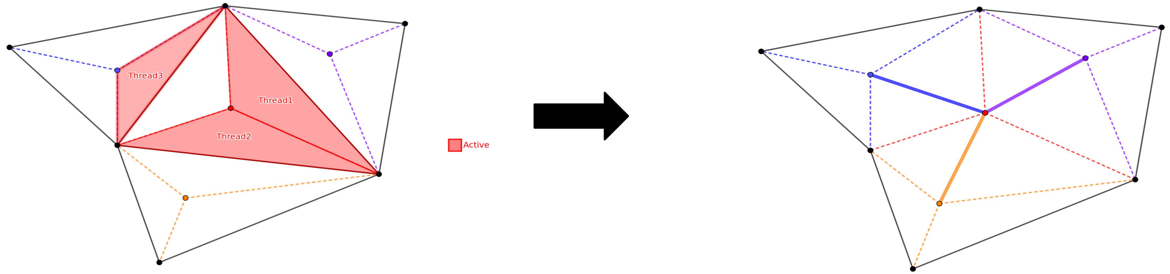


[Figure 7: Compaction lists of active triangle]

Step 2) Pre-processing for Edge Flipping: Marking active triangles.

- Mark only one triangle among two triangles incident to each edge of original triangles (triangles before point insertion).
- Can be done in parallel by assigning a thread to each edge.
- Note that edge-flip is not required between two triangles incident to edge added by point insertion.

Algorithm Details: (2) gDel2D (Cao et al.)

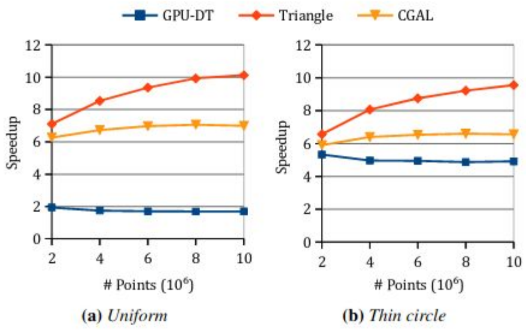


[Figure 8: Parallel edge flipping on active triangles]

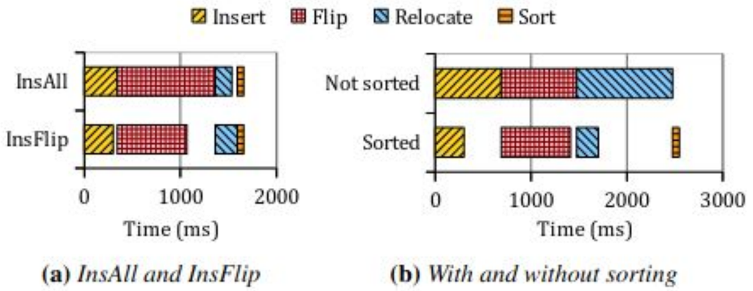
Step 3) Parallelizing Edge Flipping.

- Assign threads only for active triangles.
  - Avoid dual checkings.
  - Well covering every case.
- Relocate & sort points.

Algorithm Details: (2) gDel2D (Cao et al.)



[Figure 9: Speed up of *gDel2D* over *GPU-DT*, *Triangle* and *CGAL*]



[Figure 10: Time breakdown of *gDel2D*]

Performance analysis of *gDel2D*

- *gDel2D* performs significantly better than other widely used algorithms.

Algorithm Overview: gCrust

Parallelize each step of *Crust* determination.

Steps of *Crust* — Parallelized / Not Parallelized

DT(S) construction → V(S) from DT(S) → DT(S U V(S)) construction → Edge determination

DT(S) construction

- Parallelized using *gDel2D*.

V(S) from DT(S)

- Can be obtained by computing the *circumcircle* of each Delaunay triangle.
- Parallelized by assigning a thread for each Delaunay triangle.

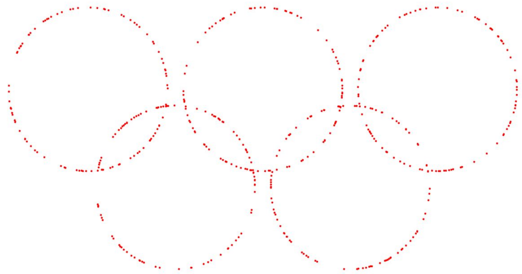
DT(S U V(S)) construction

- Parallelized using *gDel2D*.

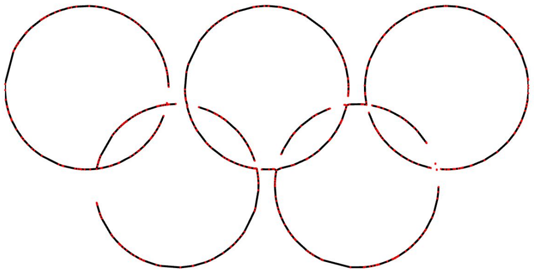
Edge determination

Artifact available!

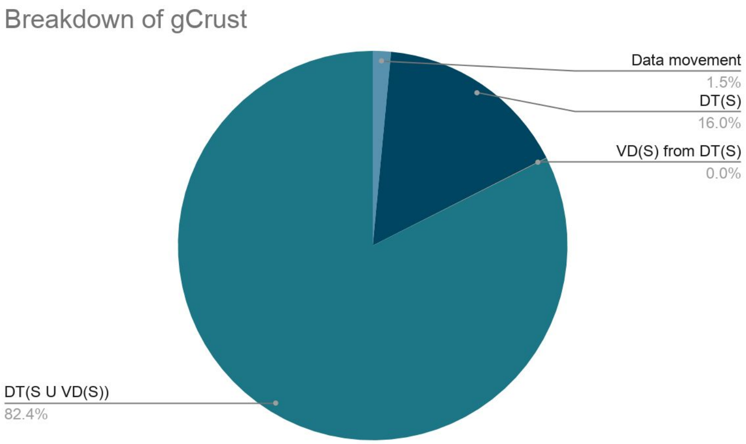
Results



[Figure 11: Input with 500 points]



[Figure 12: Output of *gCrust*]



[Figure 13: Breakdown of *gCrust* for 1M input points]

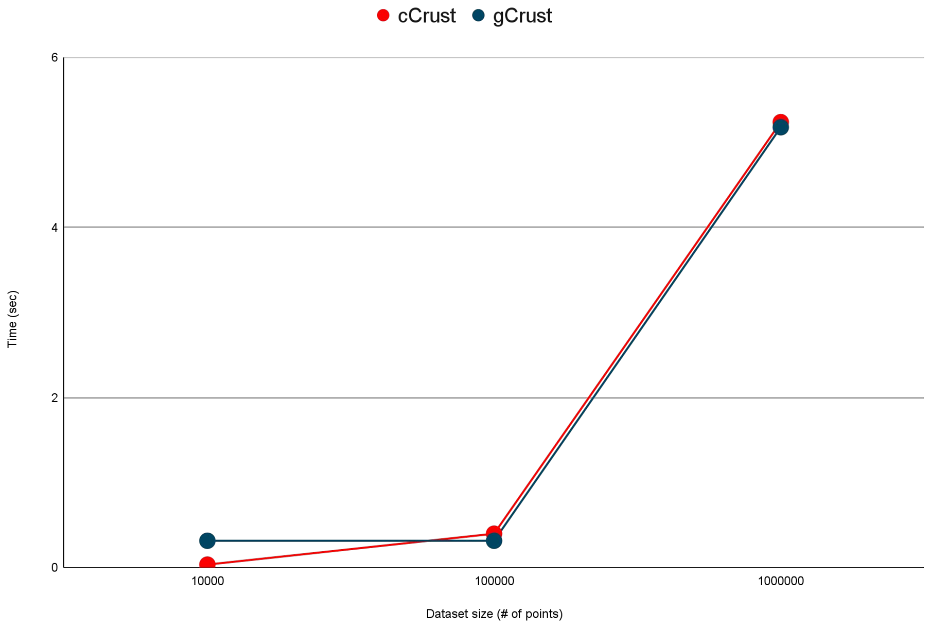
Evaluation: (1) Comparatives & Constraints

cCrust: manually implemented *Crust* algorithm with CGAL DT algorithms.

[Experimental environments]
GPU: Nvidia RTX 2060 / CPU: intel i7 2900K

Artifact available!

Evaluation: (2) Results & Analysis



[Figure 13: *gCrust* is slightly faster than *cCrust* for dataset with more than 100,000 points.]

Discussion

1) Our experimental environment was limited due to lack of budget.
   - More experiments on various GPU and CPU may provide new insights.

2) *Crust* is the simplest algorithm for 2D curve reconstruction and 2D curve reconstruction is the simplest shape reconstruction problem.
   - We may extend topic to 3D surface reconstruction or other curve reconstruction algorithms such as optimal transportation reconstruction.

Conclusion

We implemented the *gCrust*, a parallelized version of *Crust* [2] on GPU using *gDel2D* [1].

Compared to the CPU implementation of *Crust* using CGAL DT 2, it shows slightly better performance for dataset with more than 100,000 points.

The performance improvement was not as remarkable as expected. (Cao et. al. claimed that *gDel2D* performs 6 times better than *CGAL DT2* according to [1].)

References

[1] T.-T. Cao, A. Nanjappa, M. Gao, T.-S. Tan, A gpu accelerated algorithm for3d delaunay triangulation, in: 18th SIGGRAPH Symposium on Interactive 3D Graphics and Games, ACM, New York, NY, USA, pp. 47–54, 2014.

[2] N. Amenta, M. Bern, and D. Eppstein. The crust and the beta-skeleton: combinatorial curve reconstruction. Graphical Models and Image Processing 60/2:2, pp. 125–135, 1998.

[3] SHEWCHUK, J. R. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In the1st Workshop on Applied Computational Geometry. ACM, New York, NY, USA, 1996.

[4] M. Gao, T.-T. Cao, A. Nanjappa, T.-S. Tan, Z. Huang, "gHull: A GPU algorithm for 3D convex hull," ACM Transactions on Mathematical Software (TOMS), vol. 40, no. 1, pp. 1-19, 2013.

[5] Mei, G. CudaChain: an alternative algorithm for finding 2D convex hulls on the GPU. *SpringerPlus* 5, 696, 2016.

End of Document