A GPU Accelerated Algorithm for 3D Delaunay Triangulation

THANH-TUNG CAO, TODD MINGCEN GAO
TIOW-SENG TAN

National University of Singapore

ASHWIN NANJAPPA

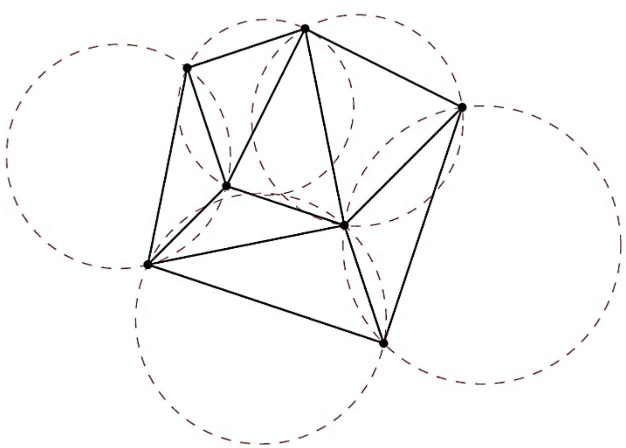
Bioinformatics Institute Singapore

Outline

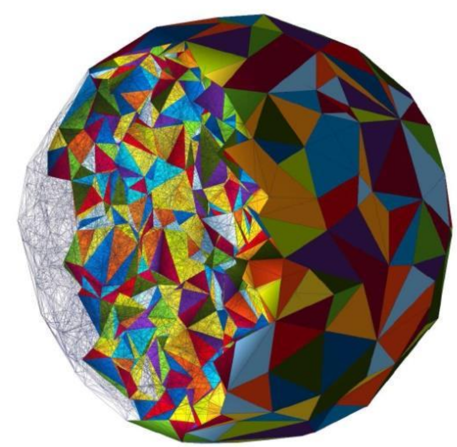
- Background
- Related work
- Algorithm
- Implementation
- Result

Background

- Delaunay Triangulation
  - Empty ball property



2D



3D

Background

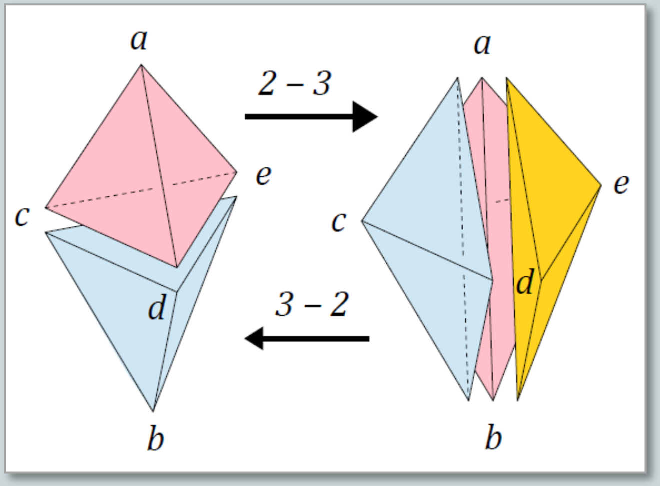
→ Related work

- Applications:
  - Graphics
  - CAD
  - Visualization
  - Scientific computation

⇒ Sequential algorithms

Related work

- Sequential algorithms:
  - Incremental construction
  - Divide-and-conquer
  - Incremental insertion
    - Points are inserted one by one
    - Triangulation is locally fixed after each insertion
      - Bowyer-Watson's algorithm [1981]
      - Flipping algorithm [Joe 1991]



Related work

- Parallel and multi-core algorithms:
  - Incremental construction → high complexity
  - Domain partitioning → GPU needs thousands of partitions

  - Incremental insertion [Batista et al. 2010]
    - Several points are inserted at a time
    - Each insertion modifies a small region
    - Conflict → Rollback

Related work

→ Algorithm

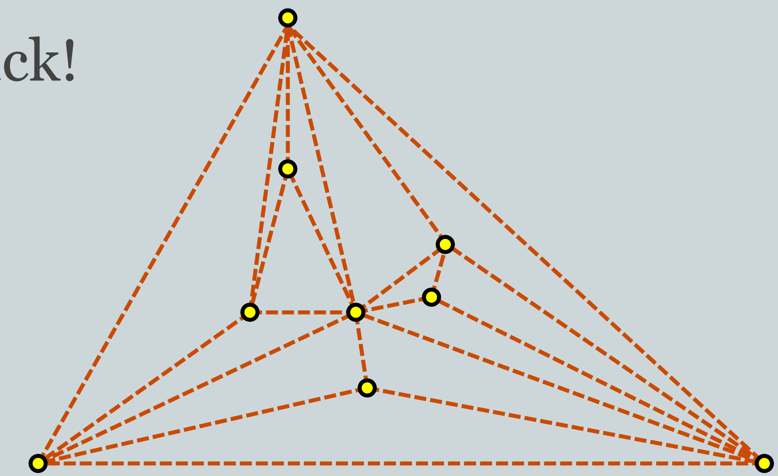
- GPU algorithms:
  - Experiment with Batista et al.'s approach
    - 1 million points
    - At most 2000-3000 points can be inserted in each round
  - Digital Voronoi diagram approach [Qi et al. 2012]
    - Dualization not working in 3D

⇒ Our algorithm

Algorithm

- Overall approach:
  - Insert points in batches, each batch at most one point is inserted into a tetrahedron
  - Use flipping to get close to the DT
  - Use star-splaying in CPU to fix [Shewchuk 2005]

  - Problem: Flipping easily gets stuck!
    - 100K points, ≈ 6,800 bad facets (non-Delaunay and unflippable).
    - Worse for real-world data.

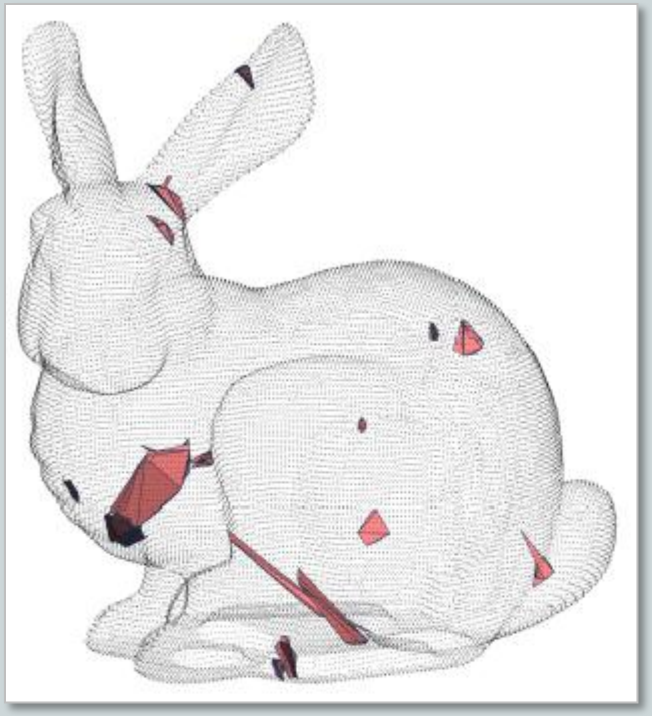


Algorithm

- Observation:
  - Do flipping after each round of point insertion

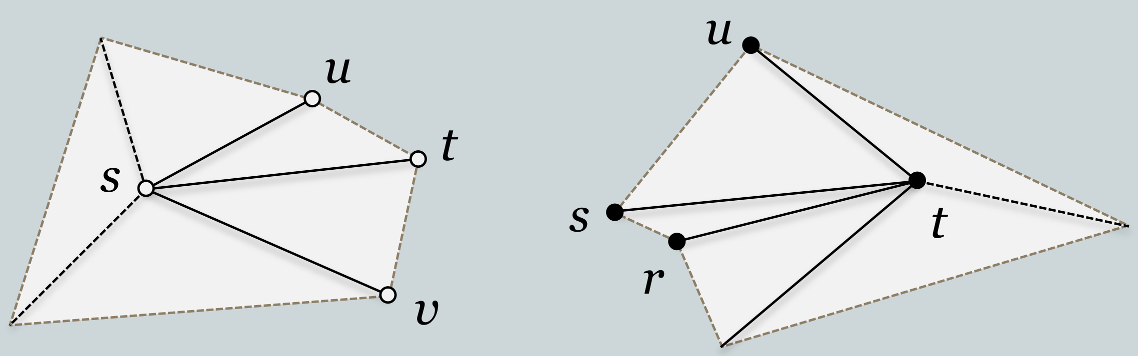
  → Much better result.

    - 100K points, 96 bad facets (vs. 6,800)
    - Bunny model (36K points), 92 bad facets
    - Now correction on CPU is acceptable.



Algorithm

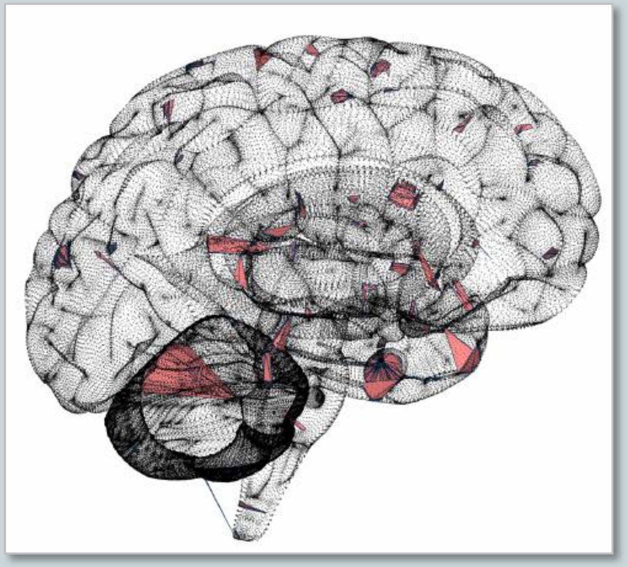
- CPU correction: star splaying algorithm
  - Lifting map: $w = x^2 + y^2 + z^2$
  - Each vertex: construct a convex star.
  - Consistency: If the star of *s* contains tetrahedron *stuv*, then the star of *t*, *u*, and *v* must also contain that tetrahedron.



2D illustration

Algorithm

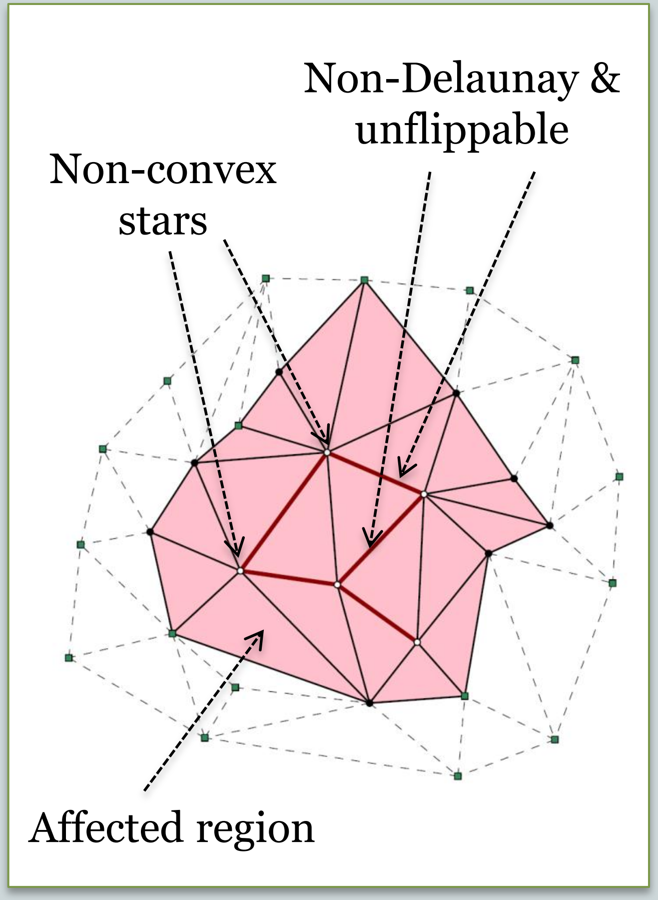
- CPU correction: star splaying algorithm
  - Difficulties: Time consuming
    - Construct convex stars (incremental insertion)
    - Check all the stars for inconsistencies
    - Convert stars back to mesh representation



⇒ Adaptive star splaying

Algorithm

- Adaptive star splaying
  - Only some small regions around the bad facets are processed
    - Construct stars incident to the bad facets.
    - Almost convex → use Flip-Flop [Gao et al. 2013]
    - Need another star → derive from the triangulation.
  - Almost output sensitive.



2D illustration

⇒ Pseudo-code

GPU Implementation


```
Initialize;

While there are points not inserted
  Pick one point per tetrahedron and insert;

  While there are modified tetrahedra
    Collect the modified tetrahedra;
    Process and identify possible flips;
    Perform the flips;
    Update location of remaining points;
```

GPU Implementation

- Thread divergence:
  - Compact the list of modified tetrahedra before processing
  - Exact predicates [Shewchuk 1996]
    - Use only fast check in 1st kernel
    - Collect those that require exact computation
    - Do the exact computation in 2nd kernel
  - Point location:
    - Store the flips, build flip history DAG
    - Trace the DAG to locate

GPU Implementation

→ Result

- Memory access:
  - Rearrange the data to improve the GPU cache efficiency.
    - Sort the input points by the Z-curve
    - Sort the tetrahedra list by the minimum vertex indices.

⇒ Experiment setting

Result

- Experiment settings:
  - CPU: Intel I7 2600K 3.4Ghz, 16GB RAM
  - GPU: NVIDIA GTX 580, 3GB VRAM
  - CUDA 5.0, VS 2012.
  - CGAL 4.2

Result

- 3D Speedup:
  - Synthetic data: Uniform, Gaussian, sphere, grid.
    - Up to 1.5 million points
    - 8-10 times faster than CGAL
  - Real models: Armadillo, Angel, Dragon, Happy Buddha…
    - 6-9 times speedup over CGAL

⇒ 2D Speedup

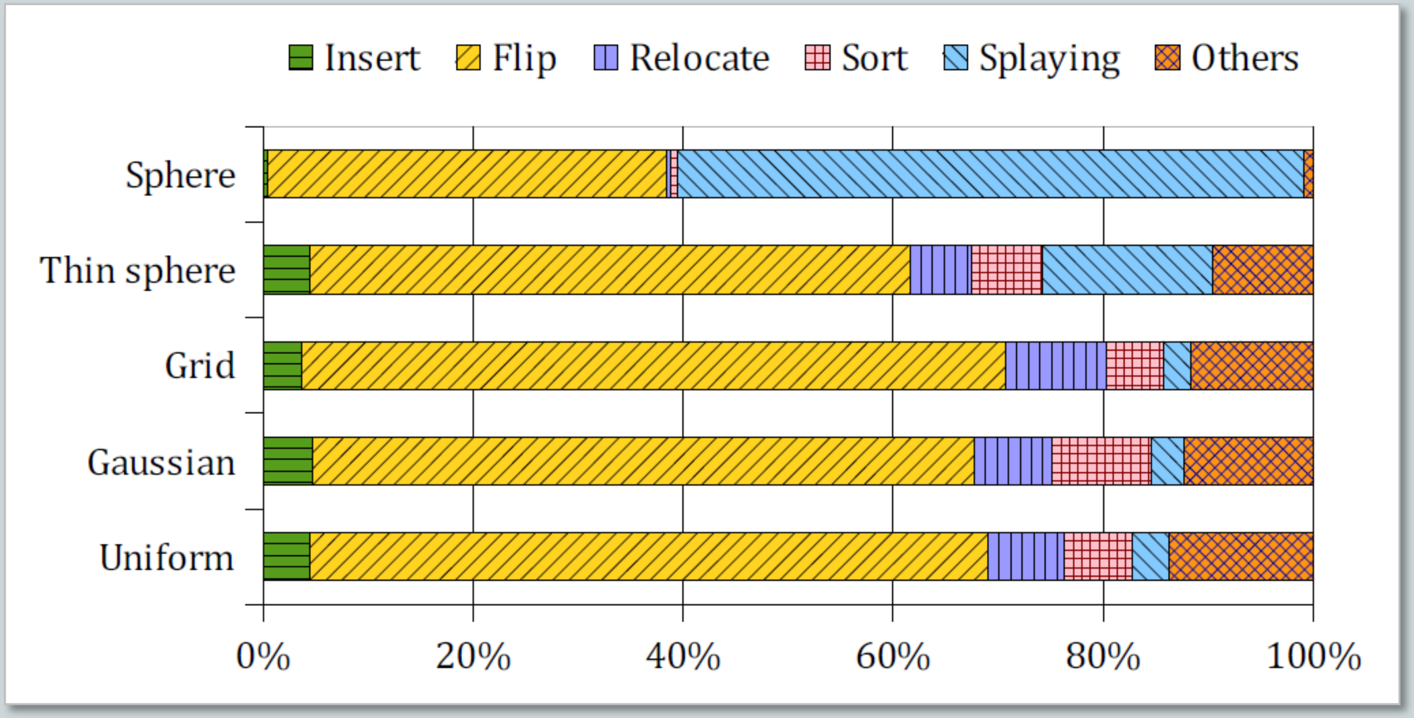
Result

- Also implement for 2D DT:
  - Synthetic data:
    - 10 times over *Triangle*, 7 times over CGAL
    - 2 times faster than [Qi et al. 2012]

⇒ Time breakdown

Result

- Time breakdown

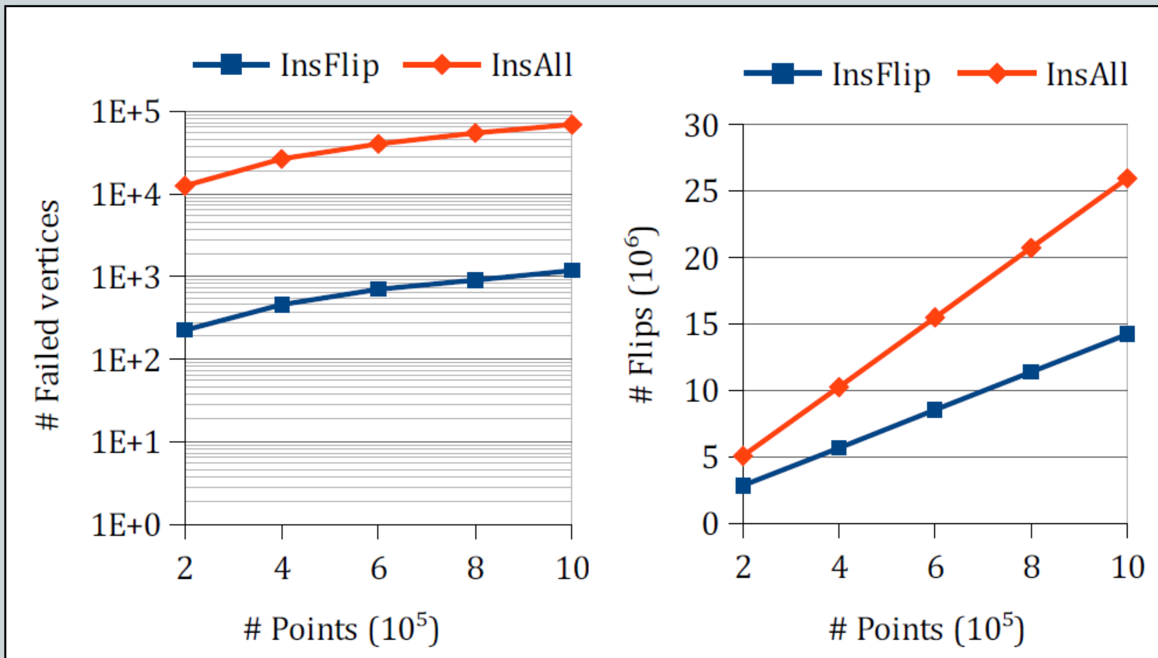


⇒ Insert-Flip vs. InsAll-Flip

Result

→ Conclusion

- Insert-Flip vs. InsertAll-Flip
  - 100 times less bad facets
  - 40% less flips



⇒ Conclusion

Conclusion

- New algorithm for DT construction on GPU
- Both 2D and 3D (possibly higher)
- Uniform and non-uniform point set
- Exact computation, robust.

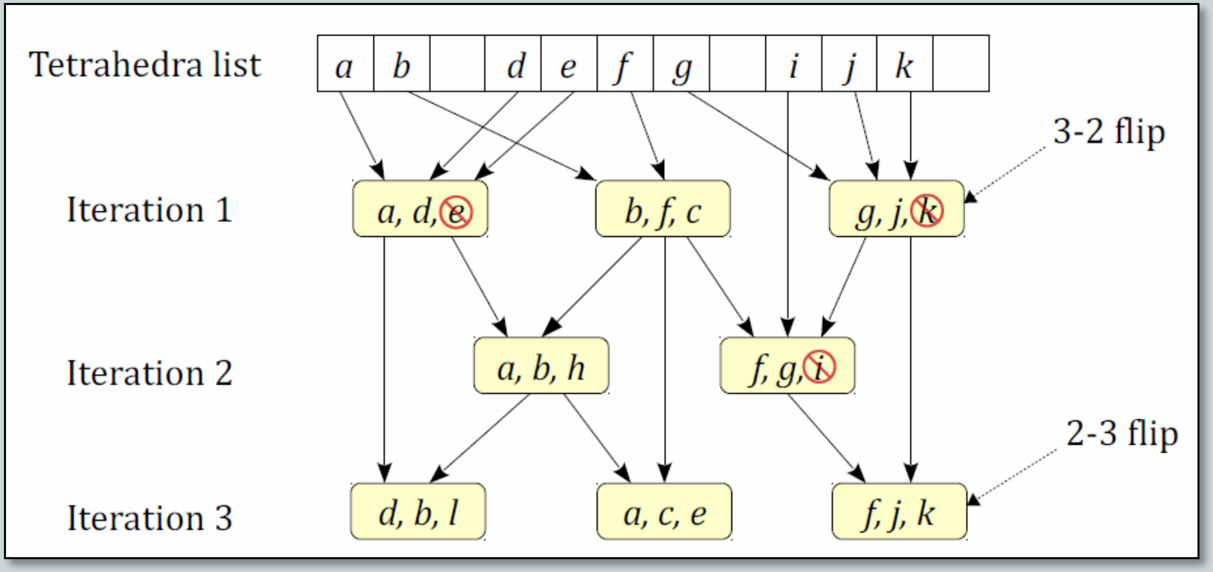
- Limitation:
  - Needs to copy the result to CPU for splaying
  - Sequential flipping on some pathological cases
  - Memory bound implementation

End

Thank you!

GPU Implementation

- Point location:
  - Store the flips in all the iterations
  - Construct the history DAG of flips
  - Update the point location at the end



⇒ Memory access

Result

- Stars involved in the CPU star splaying
  - Significantly more for non-uniform point sets
  - Still reasonably small