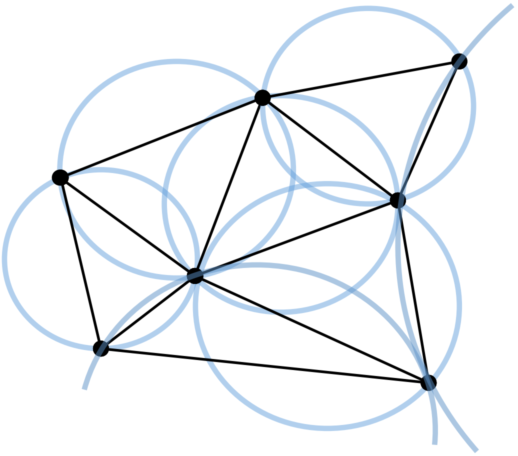
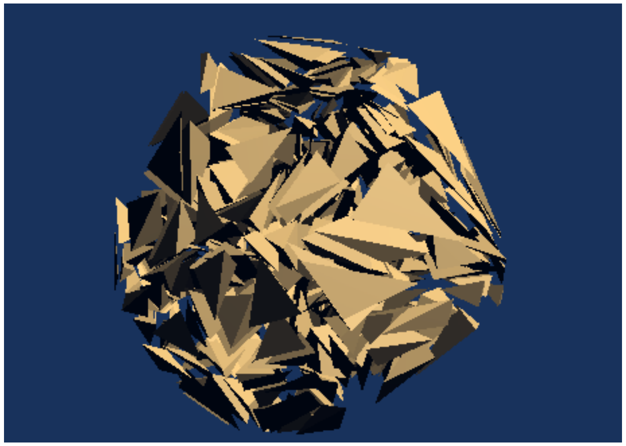
3D Delaunay Triangulation

Libin Lu, Weijing Liu, Zhuoheng Yang

Delaunay Triangulation



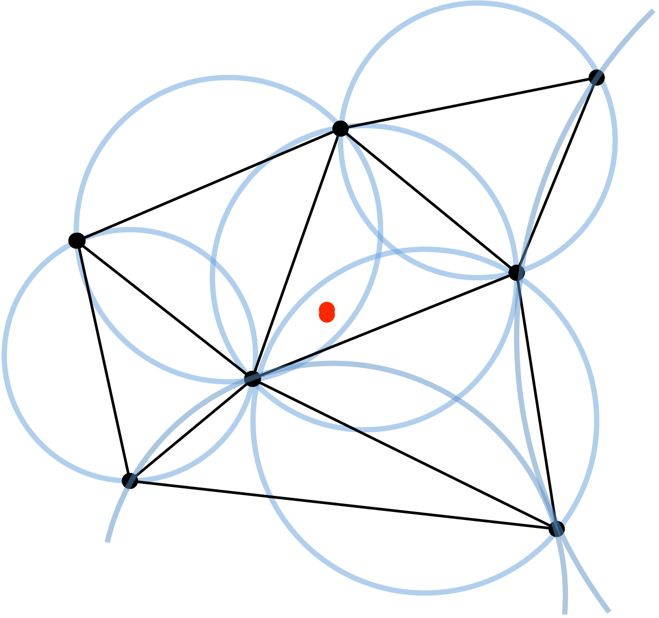


In 2-D, a Delaunay triangulation for a set P of points in a plane is a triangulation DT(P) such that no point in P is inside the circumcircle of any triangle in DT(P).[1]

In 3-D, a Delaunay triangulation for a set P of points in space is a triangulation DT(P) such that no point in P is inside the circumsphere of any tetrahedron in DT(P).

[1] http://en.wikipedia.org/wiki/Delaunay_triangulation

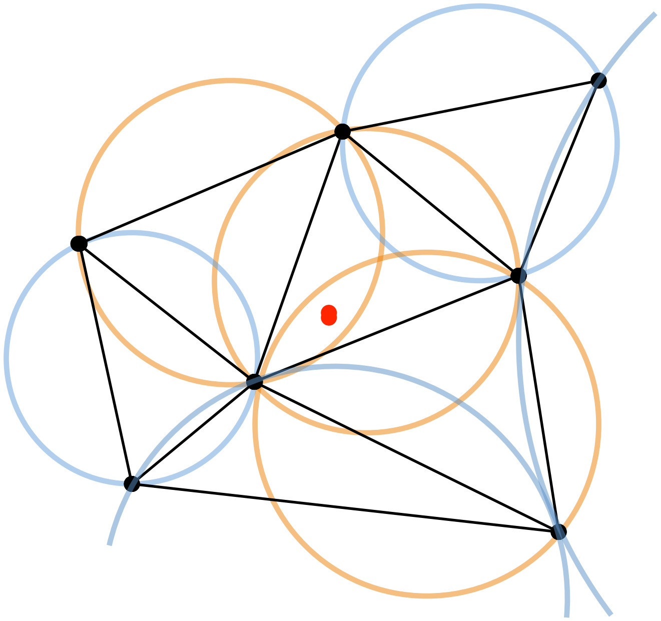
Bowyer–Watson Algorithm



The Bowyer–Watson algorithm is an incremental algorithm. It works by adding points, one at a time, to a valid Delaunay triangulation of a subset of the desired points. After every insertion, any triangles whose circumcircles contain the new point are deleted, leaving a star-shaped polygonal hole which is then re-triangulated using the new point. [2]

[2] http://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm

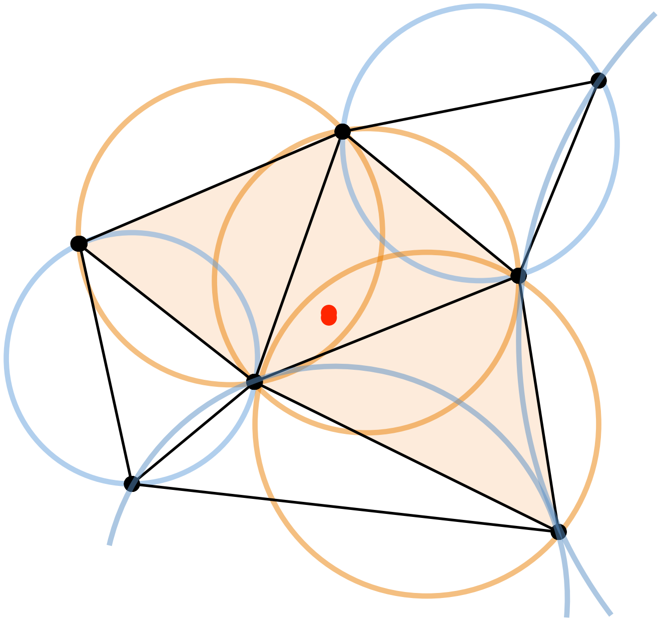
Bowyer–Watson Algorithm



The Bowyer–Watson algorithm is an incremental algorithm. It works by adding points, one at a time, to a valid Delaunay triangulation of a subset of the desired points. After every insertion, any triangles whose circumcircles contain the new point are deleted, leaving a star-shaped polygonal hole which is then re-triangulated using the new point. [2]

[2] http://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm

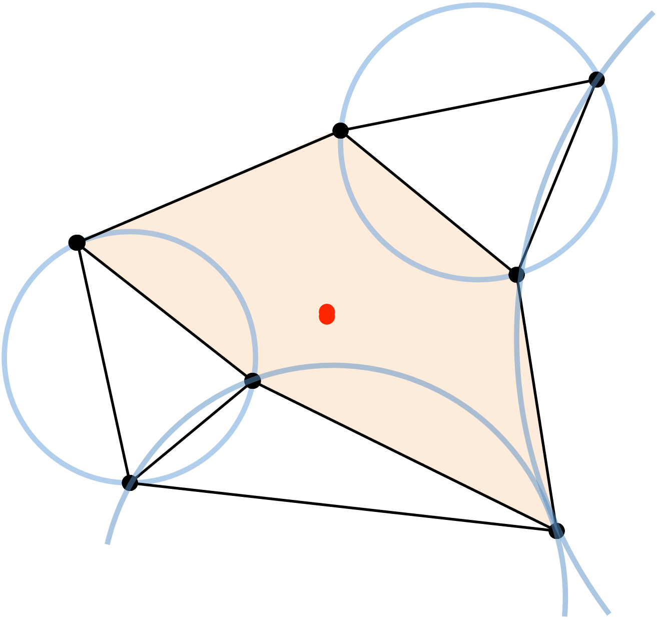
Bowyer–Watson Algorithm



The Bowyer–Watson algorithm is an incremental algorithm. It works by adding points, one at a time, to a valid Delaunay triangulation of a subset of the desired points. After every insertion, any triangles whose circumcircles contain the new point are deleted, leaving a star-shaped polygonal hole which is then re-triangulated using the new point. [2]

[2] http://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm

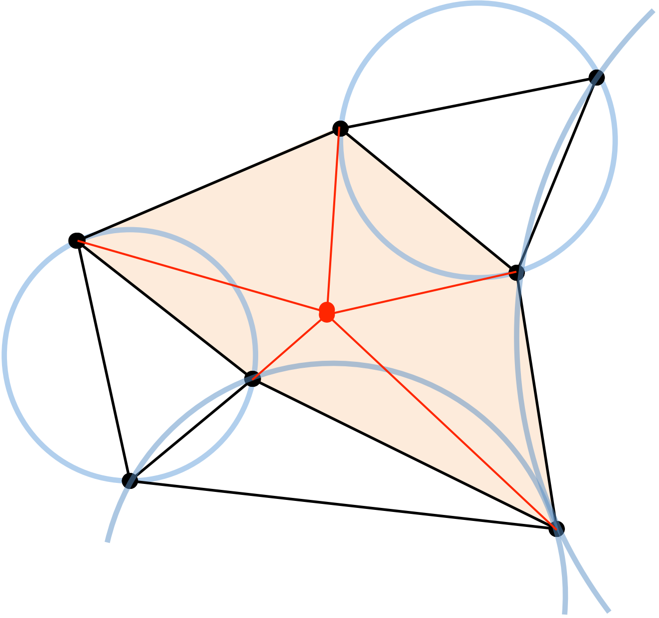
Bowyer–Watson Algorithm



The Bowyer–Watson algorithm is an incremental algorithm. It works by adding points, one at a time, to a valid Delaunay triangulation of a subset of the desired points. After every insertion, any triangles whose circumcircles contain the new point are deleted, leaving a star-shaped polygonal hole which is then re-triangulated using the new point. [2]

[2] http://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm

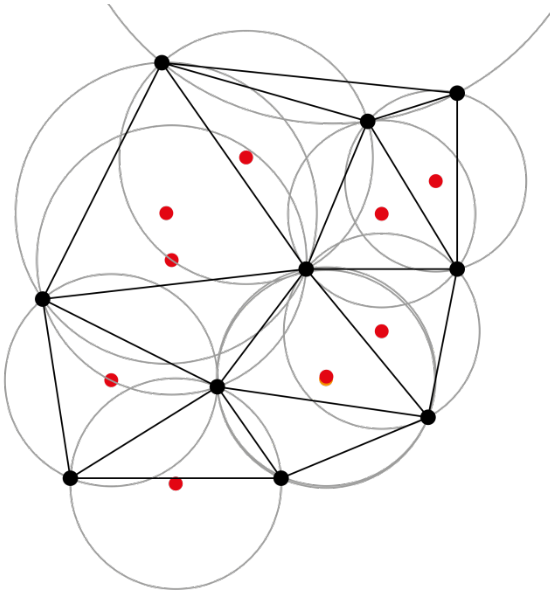
Bowyer–Watson Algorithm



The Bowyer–Watson algorithm is an incremental algorithm. It works by adding points, one at a time, to a valid Delaunay triangulation of a subset of the desired points. After every insertion, any triangles whose circumcircles contain the new point are deleted, leaving a star-shaped polygonal hole which is then re-triangulated using the new point. [2]

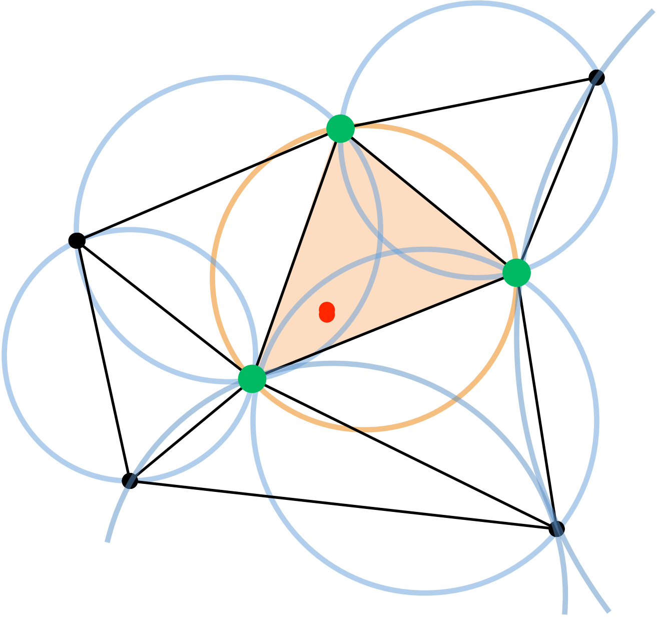
[2] http://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm

Parallelism



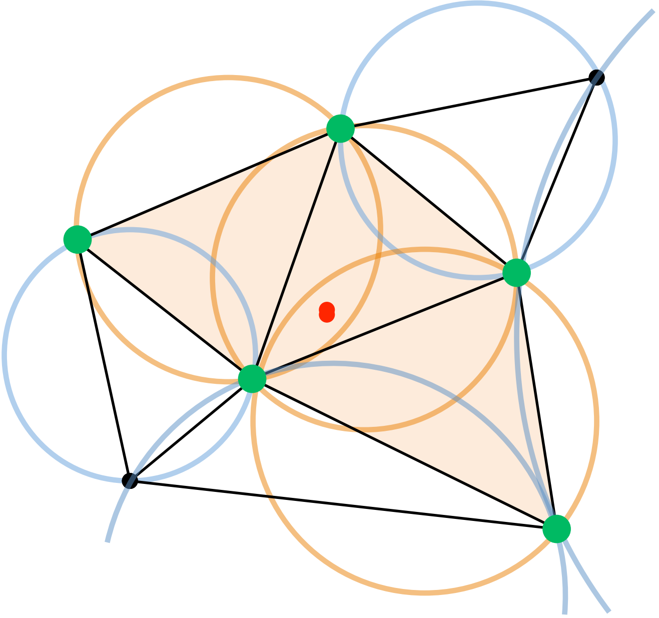
1. Each task represents a triangle in plane containing some points.
2. Each thread tries to get a task and insert points, triangles far from each other can perform insertion at the same time.

Task insertion kernel



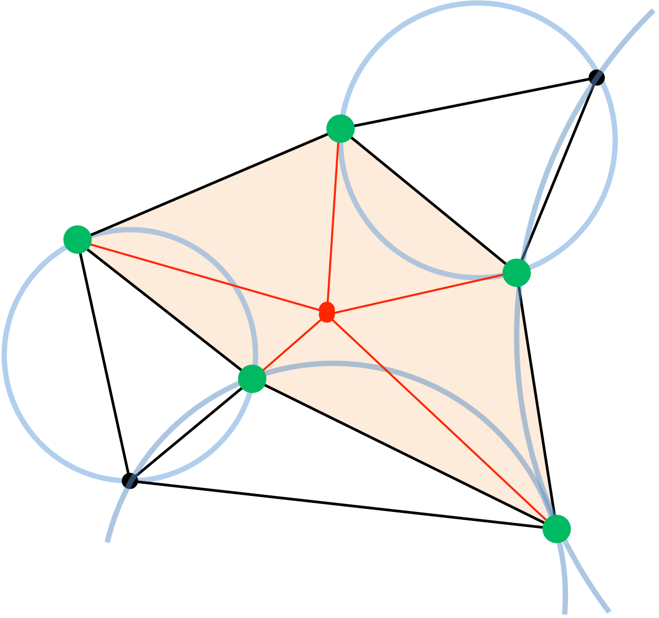
Lock the vertex of task triangle.

Task insertion kernel



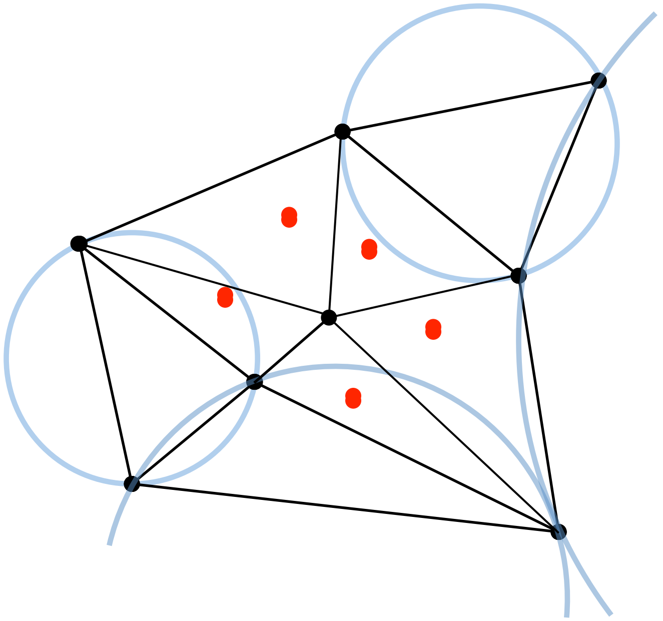
Lock more points, all the vertex of the triangles whose circumcircle contain the point to insert.

Task insertion kernel



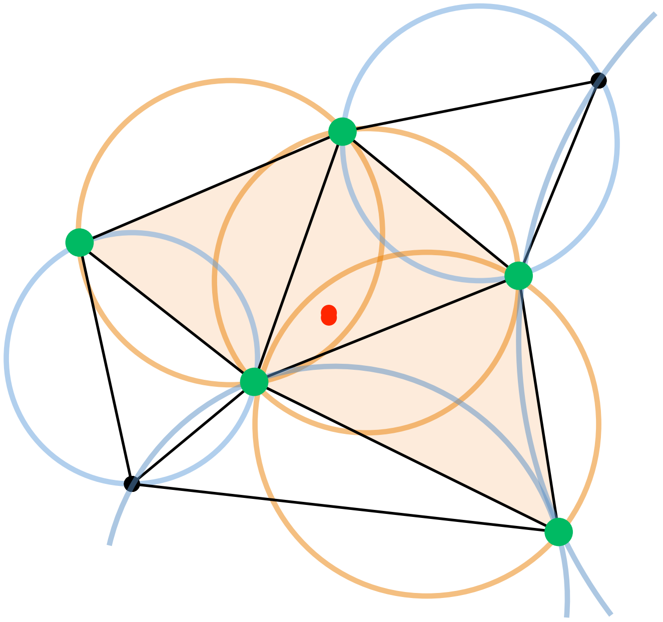
Manipulate them, create new triangles

Task insertion kernel



Sort remaining points into new triangles, put them into task queue, release the locks.

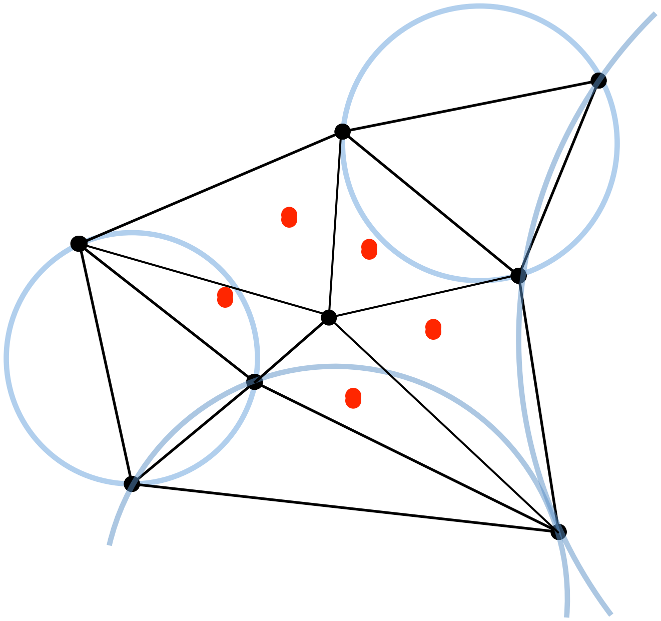
Prevent Deadlocks



Dead lock?

- If lock fail, try again later!
- Why not lock vertex in indexing order?

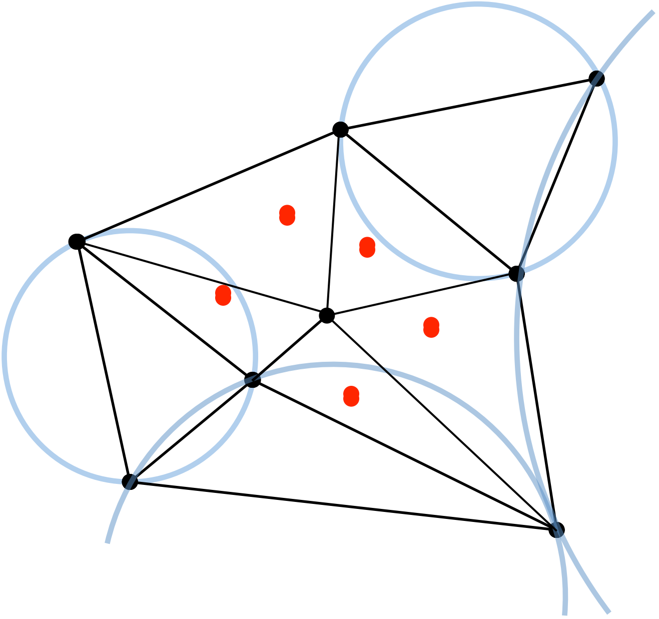
Using TBB



Tree style relations between tasks

- Not really

Using TBB

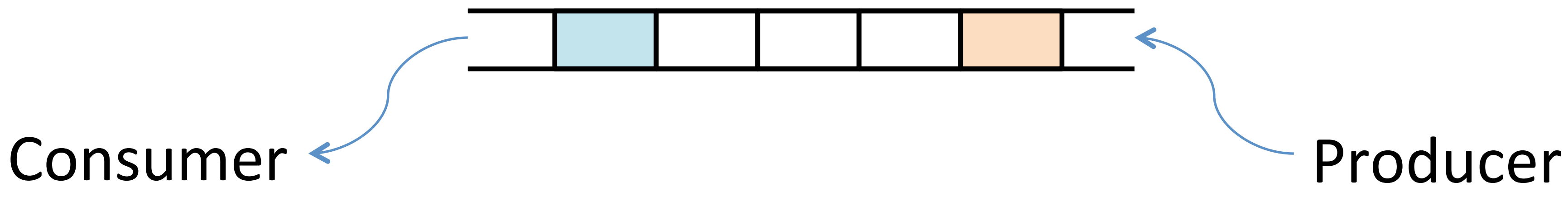


Tree style relations between tasks

- Not really

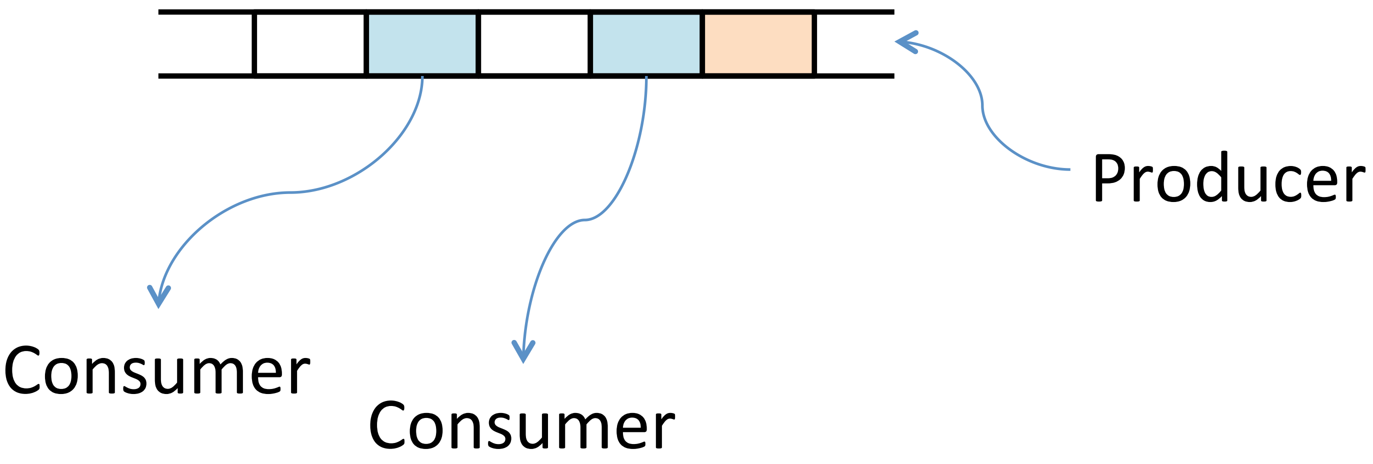
Use customized task queue, draw task randomly from queue.

Customized Task Queue



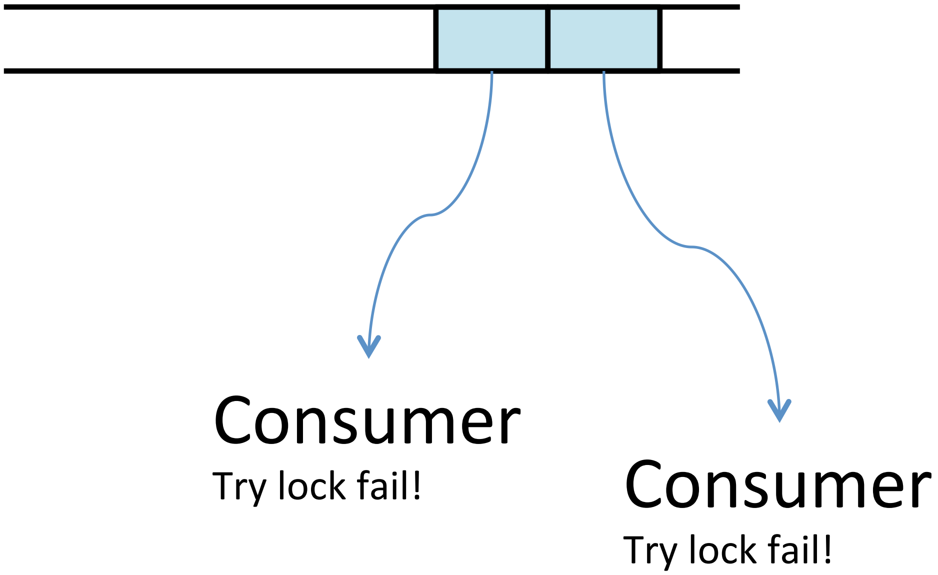
Producer-consumer problem

Customized Task Queue



Fetch a task randomly

Customized Task Queue



Consumer

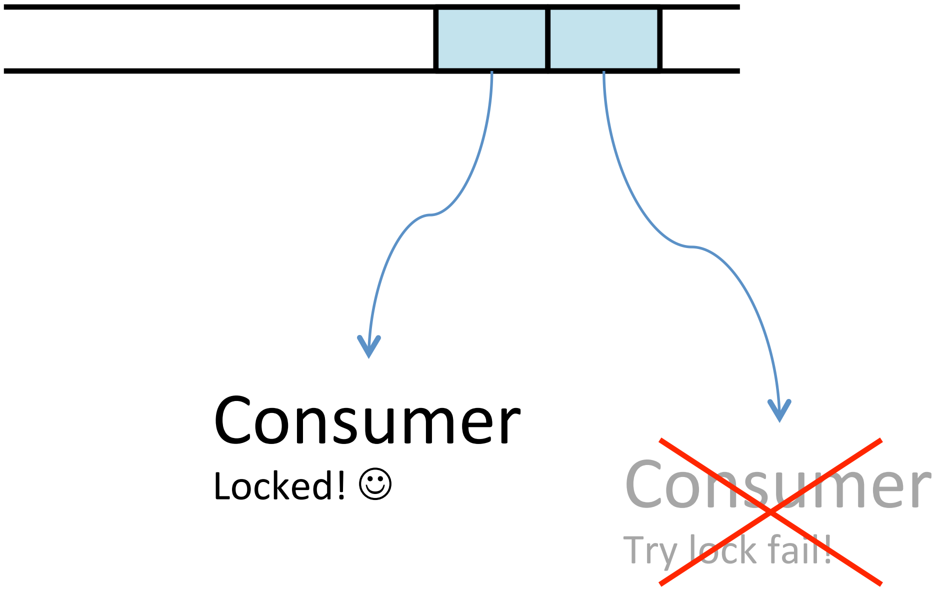
Try lock fail!

Consumer

Try lock fail!

Too many conflicts if tasks are highly related.
Starvation!

Customized Task Queue



Consumer

Locked! ☺

~~Consumer~~

~~Try lock fail!~~

Only allow one consumer if there are too few tasks.

Memory Locality

Goal:

Data associated with an insertion task should shares a small number of cache lines, which improves data locality.

Ideas:

1. Vertices that are spatially related should also stay close in memory.(done)
2. Maybe compress data(not yet)

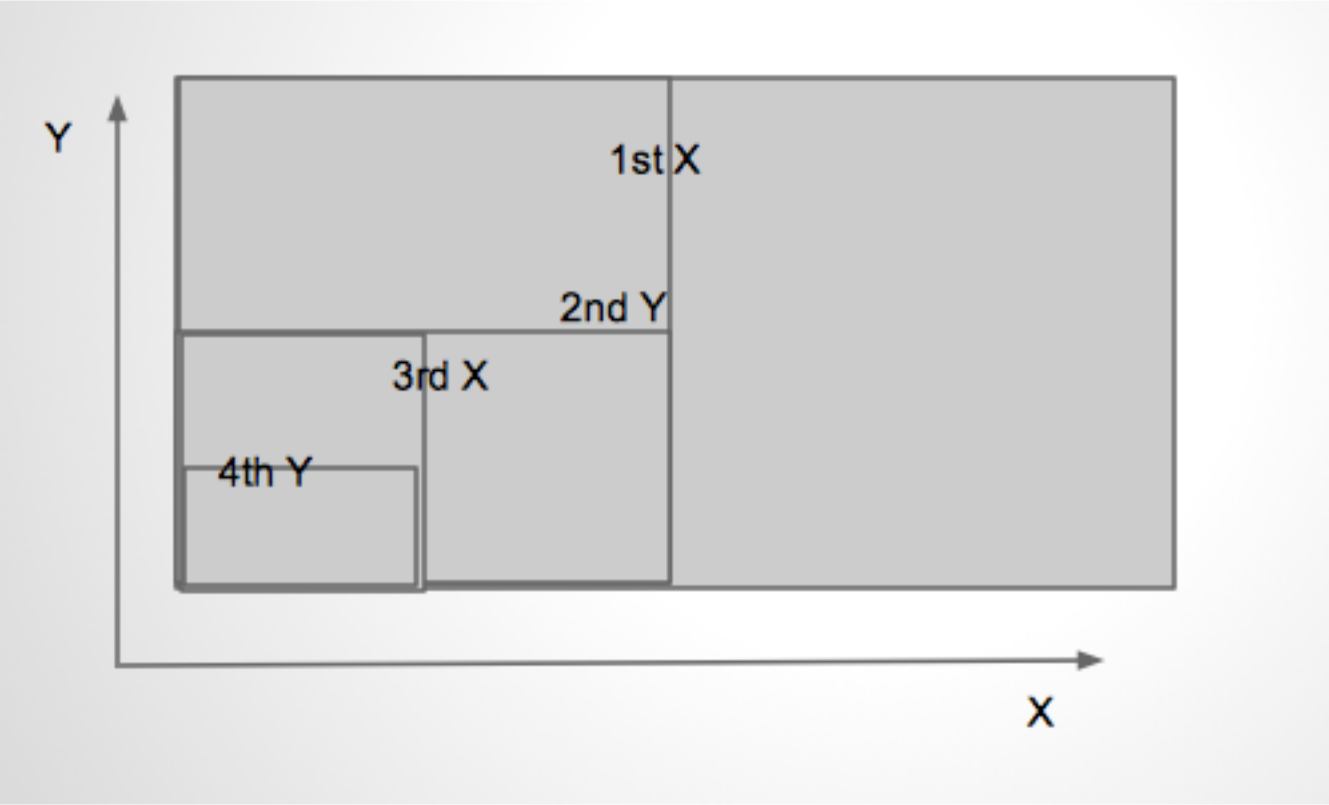
Spatial Sort

Algorithm: X-Y-Z Cuts

- 1. Find which of the x,y,z axes has the greatest diameter.
- 2. Find the approximate median(M) of D.
- 3. Partitions the points using M.
- 4. Recursively apply X-Y-Z Cuts to one side of the points first, then the other.
- (Sounds like Quicksort).

Spatial Sort

Algorithm: X-Y-Z Cuts (Example in 2D)



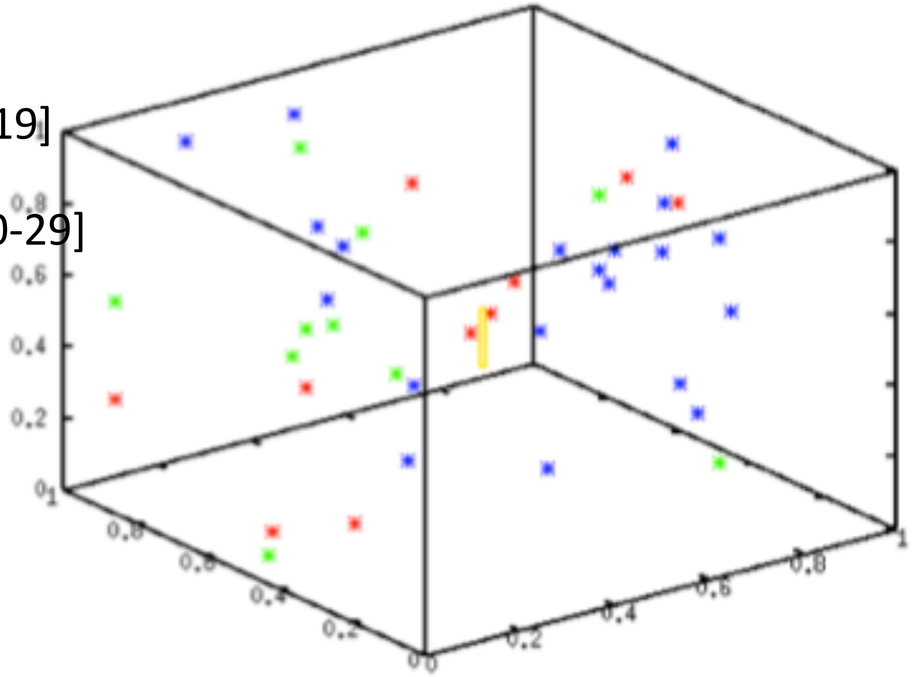
Spatial Sort

X-Y-Z Cuts (Before sorting)

Red:points[0-9]

Blue:points[10-19]

Green:points[20-29]



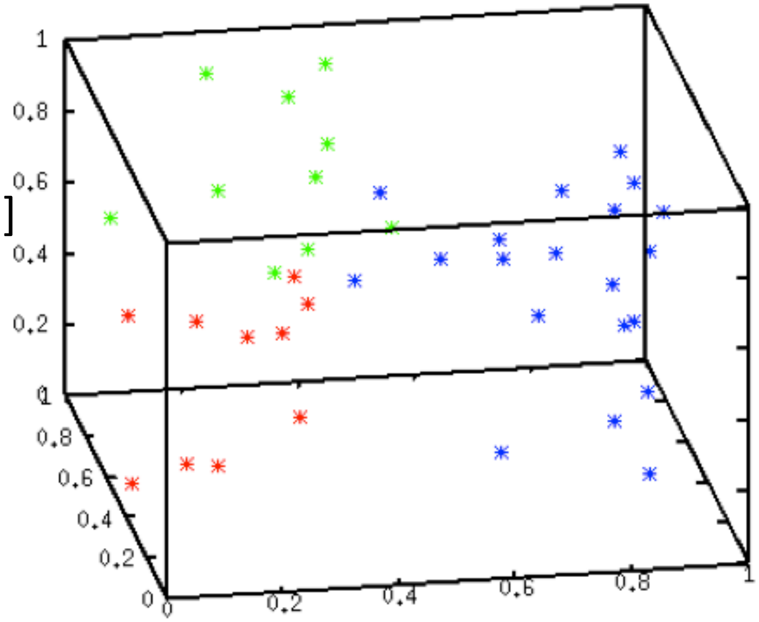
Spatial Sort

X-Y-Z Cuts (After sorting)

Red:points[0-9]

Blue:points[10-19]

Green:points[20-29]



Spatial Sort

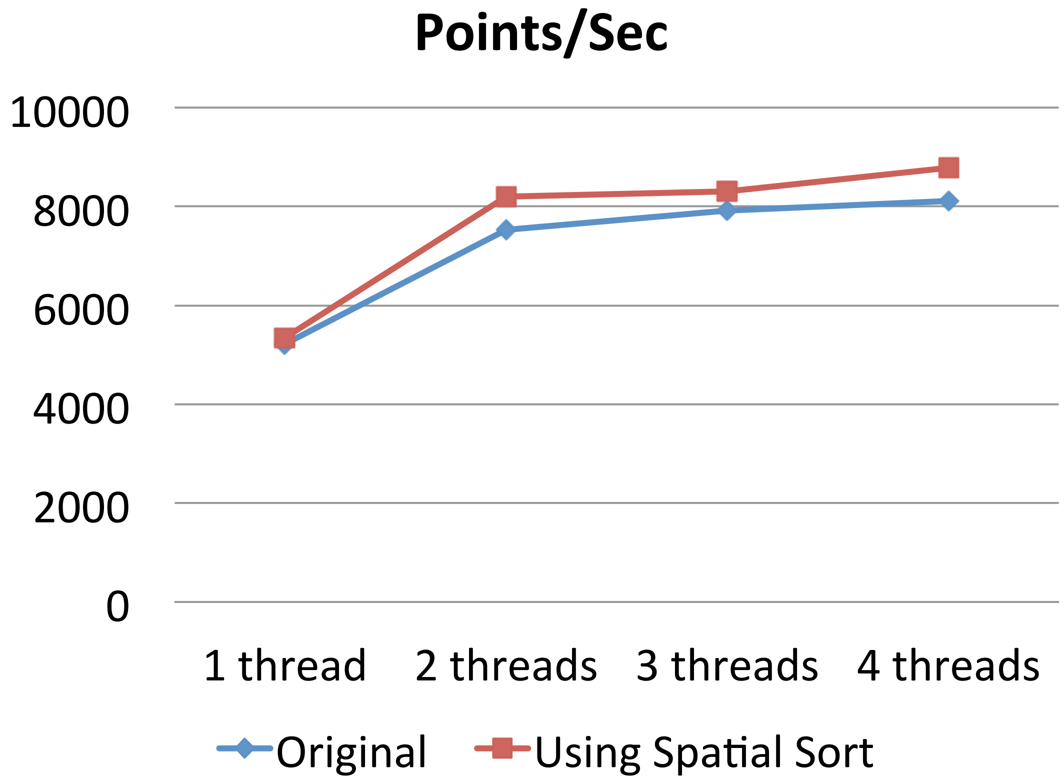
X-Y-Z Cuts Optimization

- 1. Inplace sorting. (minimize temporary memory allocation)
- 2. Parallel sorting.
- **Key**: at step #4.

✖ Recursively apply X-Y-Z Cuts to one side of the points first, then the other.

➡ Recursively apply X-Y-Z Cuts to **both** sides of the points **concurrently**.

Performance With Spatial Sort



Conclusion

Difficult to parallel 3D Delaunay triangulation
 - Because of complicated data dependencies

More difficult to achieve high efficiency
 - hard to distribute irrelative tasks to different threads