Distributed Kinetic Delaunay Triangulation

유태원° 최성희 이현익 이진원
한국과학기술원 전자전산학과 전산학전공
{yutai°, sunghee}@gclab.kaist.ac.kr, {hyonigi, jwlee}@nclab.kaist.ac.kr

Taewon Yoo° Sunghee Choi Hyonik Lee Jinwon Lee
Division of Computer Science, Department of Electrical Engineering and Computer Science
Korea Advanced Institute of Science and Technology

Abstract

This paper proposes a distributed algorithm to maintain the Delaunay triangulation of moving points. We assume that every point is a processor which can only communicate with the adjacent points connected by edges in the Delaunay triangulation. The topology changes of the Delaunay triangulation due to the movement of the points are updated automatically by local operations of the points without any centralized processor or global information.

1. Introduction

Massively Multiplayer Online Games(MMOG) have become one of the most important and popular Internet applications. Since a MMOG player mainly interacts with adjacent players in a virtual space, the player has to know which players are nearby. Currently this proximity information in the virtual world is given from the server. This is because traditionally MMOG has client-server architecture. Therefore the overall scalability is strictly limited by the capacity and computation power of the server.

To overcome this limitation, peer-to-peer overlay network is suggested as an alternative architecture for MMOG. On a peer-to-peer environment, a node has to maintain the location information of nearby players by itself. And the location information has to be reconstructed dynamically because proximity can be changed by the movement of players. The Delaunay triangulation has been used for overlay network since by definition it connects nearby points with edges. Though there have been several algorithms to construct the Delaunay triangulation locally for peer-to-peer applications, no paper has addressed the problem of maintaining the Delaunay triangulation of moving nodes directly in a distributed setting.

In this paper, we propose an algorithm to maintain the distributed kinetic Delaunay triangulation. Here the term *distributed* means the computation and maintenance of the Delaunay triangulation is done locally by each point (or node) of the triangulation. And the term *kinetic* means that points (nodes) are allowed to move and the algorithm can handle the topological changes according to the motion of points. To the best of our knowledge, this is the first paper to address the problem of maintaining Delaunay triangulation of continuously moving points in a distributed setting.

2. Related Works

The Delaunay triangulation is a fundamental problem in computational geometry and has been a major research topic in the community. The kinetic Delaunay triangulation problem is also studied but only in a centralized setting.

Roos[1] proposed a mechanism to construct and maintain the Delaunay triangulations of moving points in 2D. They store all the calculated next topological events of Delaunay triangulations in a centralized queue, and then update the triangulation according to the time sequence of the events. Albers et al.[2] extend this for kinetic Delaunay triangulation in general dimension.

In the system community, the problem of constructing 2D Delaunay triangulations in a distributed way has been studied for peer-to-peer applications. Liebeherr and Nahas[3] generate Delaunay-based overlay network using angular feature in two dimensional space. In [5], Simon et al. extend it to d-dimensional space using in-hypersphere-test based on greedy walk. Ohnish et al.[4] construct the Delaunay triangulation locally using flip operations. Though they remark that they can handle movements of points, it is by using insertions and deletions of points. None of the works deals with the continuous movement of points directly in a distributed approach.

3. Backgrounds

In this section, we briefly introduce several definitions and properties related to the Delaunay triangulation.

A *triangulation* of a vertex set V is a decomposition of the convex hull of V into triangles by maximal set of non-intersecting diagonals.

An edge is defined as *Delaunay* if it has any circumscribing circle which contains no vertex of V.

The *Delaunay triangulation* D of V is defined as a triangulation whose edges are all Delaunay edges.

We can compute the Delaunay triangulation D of V using a local property of the Delaunay triangulation.

An edge e is *locally Delaunay* if

(1) e is a convex hull edge of T or

(2) e is an internal edge of T and the circumscribing circle of an adjacent triangle of e does not contain the other adjacent triangle of e.

The following theorem guarantees that we can compute the Delaunay triangulation of V by only maintaining every edge to be locally Delaunay.

Theorem 1 [6] A triangulation T whose edges are all *locally Delaunay* is a *Delaunay triangulation.*

A locally Delaunay edge *e'* is created from a non-locally Delaunay edge *e* by flipping. Flipping an edge *e* means to delete *e* and insert the other diagonal edge of quadrilateral which is combined by two triangles adjacent to *e*. The following lemma shows that flipping produces a locally Delaunay edge from a non-locally Delaunay edge.

Lemma 1 [6] Let *e* be an edge of a triangulation of V. Either *e* is *locally Delaunay*, or *e'* created by flipping *e* is *locally Delaunay.*

We can compute the Delaunay triangulation of vertex set V using lemma 1. The Delaunay triangulation D of a triangulation T is obtained by flipping edge *e* that is not Delaunay until all the edges are Delaunay[6]. This method is called *flip algorithm* and has $O(n^2)$ computation complexity.

Roos[1] first solved the Delaunay triangulation of moving points. He proved that only two kinds of topological changes of the Delaunay triangulation occur according to the continuous movement of the points. Those topological changes occur at the moment of cocirculairty of four points or collinearity of three consecutive convex hull points.

Here we call the first class of topological events as *flip events* and the second class of topological events as *collinear events.* To handle these topological events, Roos pre-computes the event times of all the Delaunay triangulation edges and store them in the *SWAP* tree according to the temporal order of their event times. And a centralized event handler processes the topological event with the smallest event time. Thereafter it updates the SWAP tree since there are some edges whose event time is affected by the previous topological changes.

4. Algorithm

In this section we describe the assumptions and our algorithm. Our algorithm is based on Roos' approach but modified for a distributed setting.

4.1 Assumptions

We introduce the following assumptions to make our approach simple.

(1) Each node can query the location and velocity information of nodes which are connected by an edge.
(2) The clocks of nodes are synchronized.
(3) Every node moves in a bounded constant speed and when it changes its direction and speed, it notifies its adjacent nodes.
(4) Every node moves only in a triangle whose vertices are three stationary nodes.

4.2 Data structures

To handle the topological events in distributed manner, each node maintains two kinds of data structures:

- *Adjacency List:* Every node has the direct connection with nodes which are connected by Delaunay edges. This information is maintained using its own adjacency list. Adjacency list is sorted by counter-clockwise order.
- *Event Priority Queue:* Every node has its own event priority queue. The element of the queue consists of three fields: the edge which is managed by the node, the event time of that edge, and lock field(true or false). If the value of the lock field is true, the node can not trigger the corresponding event and waits until the lock is released. The elements of the queue are sorted by temporal order of the event time.

4.3 Algorithm

In our algorithm, the event of an edge *e* is handled by the randomly chosen node *v* between two nodes incident to *e*. We define the node *v* as the *event handler* of *e*. The event handler *v* of *e* firstly computes the event time of *e* and it handles the topological change on that event time of *e*. And *v* updates all the event times of edges which are affected by that topological change. When a node changes its velocity, it notifies its adjacent nodes and the node and its neighbors recalculates the event times and updates their event queues.

Now we will explain the event handling procedure.

By the assumption (4) which prevents the change of convex hull, we do not need to handle the collinear events. So, we only need to process flip events. The event handler of an edge modifies the topology of the Delaunay triangulation due to flipping and updates the event time of edges which are affected by this flip event. Since a flip event of an edge destroys 4 quadrilaterals and creates new 4 quadrilaterals-see figure 1, it is enough to re-compute the event times of 4 corresponding diagonal edges. Then, it randomly selects the event handler of newly created edge. The newly selected event handler re-computes the event time of the new edge and updates its event priory queue. The following algorithm explains the whole procedure of the flip event of edge p_ip_j by its event handler p_i.

(Notations)
p_ip_j : the edge between p_i and p_j.
p_i : the event handler of p_ip_j
p_kp_l : the newly created edge by flipping p_ip_j
$p_ip_kp_jp_l$: quadrilateral which has p_ip_j as one of its diagonals.

Algorithm Flip event handling
1. p_i waits if the lock field of the event p_ip_j is *true.*
2. p_i sends the messages to p_j, p_k, p_l to set the lock field of the event of p_ip_k, p_ip_l, p_jp_k, p_jp_l *true.*
3. p_i sends the message to p_j to remove p_i from the adjacency list of p_j.
4. p_j removes p_j from its adjacency list.
5. p_i pops the top element from its event priority queue.
6. p_i sends the messages to p_j, p_k, p_l to re-compute the event time of p_ip_k, p_ip_l, p_jp_k, p_jp_l.
7. p_i randomly selects the event handler of p_kp_l between p_k and p_l.
8. The selected event handler computes the next flip event time of p_kp_l and inserts this new event in its event priority queue.
9. p_i sends the messages to p_k and p_l to insert each other in their adjacency list.
10. p_i sends the messages to p_j, p_k, p_l to set the lock field of the event of p_ip_k, p_ip_l, p_jp_k, p_jp_l *false.*

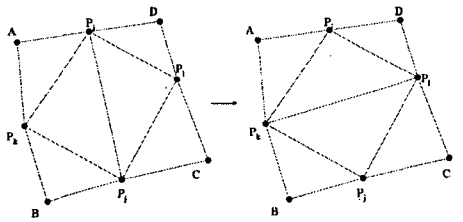

Figure 1. Flipping the edge p_ip_j destroys 4 quadrilaterals-$Ap_kp_jp_i$, $Bp_jp_ip_k$, $Cp_ip_jp_l$, $Dp_jp_lp_i$ and creates 4 quadrilaterals-$Ap_kp_lp_i$, $Bp_lp_ip_k$, $Cp_lp_kp_j$, $Dp_lp_kp_l$

4.4 The correctness of the algorithm

The correctness of our algorithm is basically based on that of the centralized algorithm of Roos[1]. However in the distributed processing environment, concurrent edge flipping may result in the *edge crossing problem* which is illustrated in Figure 2. The following lemma shows when exactly this problem takes place.

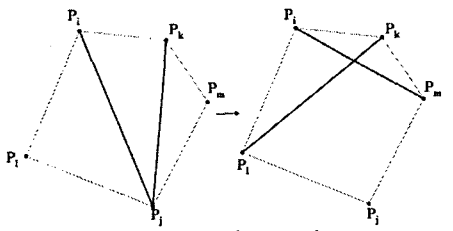

Figure 2. Concurrent flipping adjacent edges can cause the edge crossing problem.

Lemma 2 When an edge e is being flipped, the edge crossing problem occurs only by the simultaneous flip events of edges of the quadrilateral whose diagonal is e.

Proof)

The proof consists of two parts.

Claim (1) The simultaneous flip events of edge e and an edge of the quadrilateral whose diagonal is e always cause the edge crossing problem.

Assume that p_ip_j and p_kp_j in Figure 2 are being flipped concurrently. It implies that p_i, p_j, p_k, p_l, p_m lie on the same circle. Thus pentagon $p_ip_lp_jp_mp_k$ is convex. And flipping p_ip_j and p_kp_j produce p_kp_l and p_ip_m, respectively. Since these two new edges are diagonals of the convex pentagon which do not share any vertex of that pentagon, they must intersect each other.

Claim (2) the simultaneous flip events of edge e and the edge which is not an edge of the corresponding quadrilateral of e does not cause the edge crossing problem.

Since these edges are the diagonals of two non-overlapping quadrilaterals, their corresponding newly created edges do not intersect each other.

Lemma 2 shows that we can avoid the edge crossing problem induced by flipping edge e by preventing the flip events of its corresponding quadrilateral edges only and that we do not need to worry about the other edges. The steps 1, 2, 10 of our algorithm implement this using locks. Thus, the proposed algorithm correctly maintains the topology of the Delaunay triangulation under our assumptions.

5. Results

We simulated the distributed setting by implementing each node as an object with its own data structure and procedure using C++ and OpenGL. Our implementation takes the number of points as input. The location, the moving direction and the speed of each point are determined randomly. The simulation shows that our algorithm changes gracefully the topology of the Delaunay triangulation according to the linear motion of nodes during the simulation.

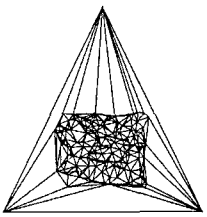

Figure 3 Screen shot of our implementation

6. Future Works and Conclusion

In this paper we proposed an algorithm to maintain the distributed kinetic Delaunay triangulation. We also implemented our algorithm. It can be used for the peer-to-peer adjacency management of MMOG.

We conclude by remarking several open issues:

Load balancing: If a node handles a lot of edge flip events, it would suffer from the event handling overheads. We want to minimize the number of events that a node handles.

Delay: In real P2P overlay network environment, the message exchanges between nodes may take quite a while due to delays. This may cause inconsistency problems. We need to make our algorithm more robust to the communication delays between nodes.

Fault-tolerance: Though insertions and deletions of nodes can be handled using existing methods, nodes may crash without any warning, in real peer-to-peer applications. It would be nice to be able to handle these cases gracefully.

References

[1] Thomas Roos, "*Voronoi Diagrams over dynamic scenes*", Discrete Applied Mathematics 43 pages 243-259, 1993.

[2] Gerhard Albers, Leonidas J. Guibas, Joseph S.B. Mitchell, Thomas Roos, "*Voronoi Diagrmas of Moving Points*", 1995.

[3] Liebeherr, M. Nahas, "*Application-layer multicasting with Delaunay triangulation overlays*", Technical Report: CS-2001-26 IEEE Journal on Selected Areas in Communications, 2001.

[4] Masaaki Ohnishi, Ryo Nishide, Shinichi Ueshima, "*Incremental Construction of Delaunay Overlaid Network for Virtual Collaborative Space*", The third international Conference on Creating, Connecting and Collaborating through Computing, 2005.

[5] Gwendal Simon, Moritz Steiner, Ernst Biersack, "*Distributed Dynamic Delaunay Triangulation in d-Dimensional Spaces*", 2005.

[6] Jonathan Richard Shewchuk, "*Lecture Notes on Delaunay Mesh Generation*", 1999.