Efficient Triangulation for P2P Networked Virtual Environments

Eliya Buyukkaya, Maha Abdallah

To cite this version:

Eliya Buyukkaya, Maha Abdallah. Efficient Triangulation for P2P Networked Virtual Environments. Multimedia Tools and Applications, 2009, Special Issue on Massively Multiplayer Online Games (MTAP-MMO), 45 (1-3), pp.291-312. 10.1007/s11042-009-0301-0 . hal-00589734

HAL Id: hal-00589734
https://hal.science/hal-00589734
Submitted on 1 May 2011

Efficient Triangulation for P2P Networked Virtual Environments

Eliya Buyukkaya and Maha Abdallah

LIP6, University of Paris 6

104, Avenue du President Kennedy

75016 Paris, France

{Eliya.Buyukkaya, Maha.Abdallah }@lip6.fr

Phone +33-1-44.27.87.{47, 93}

Fax +33-1-44.27.70.00

Abstract Peer-to-peer (P2P) architectures have recently become a popular design choice for building scalable Networked Virtual Environments (NVEs). In P2P-based NVEs, system and data management is distributed among all participating users. Towards this end, a Delaunay Triangulation can be used to provide connectivity between the different NVE users depending on their positions in the virtual world. However, a Delaunay Triangulation clearly suffers from high maintenance cost as it is subject to high connection change rate due to continuous users' movement. In this paper, we propose a new triangulation algorithm that provides network connectivity to support P2P NVEs while dramatically decreasing maintenance overhead by reducing the number of connection changes due to users' insertion and movement. Performance evaluations show that our solution drastically reduces overlay maintenance cost in highly dynamic NVEs. More importantly, and beyond its quantitative advantages, this work questions the well accepted Delaunay Triangulation as a reference means for providing connectivity in NVEs, and paves the way for more research towards more practical alternatives for NVE applications.

Keywords *Networked Virtual Environments, Peer-to-Peer Systems, Delaunay Triangulation*.

1 Introduction

Networked Virtual Environments (NVEs) are 3-D virtual worlds in which a huge number of participants play roles, and interact with their surroundings and each other through virtual representations called *avatars*. Several application areas for NVEs exist, the most popular of which are Massively Multiplayer Online Games (MMOGs) where hundreds of thousands of concurrent players are currently reported. This challenges the scalability of currently employed client-server architectures where the only way to cope with an ever growing user population is to employ more dedicated servers. Clearly, this solution is extremely expensive to deploy, and its scalability is limited by server capacity which can become a bottleneck during peak loads.

This has recently led to focusing on peer-to-peer (P2P) architectures as an alternative design choice for building scalable NVEs. By aggregating and sharing users' resources, P2P architectures achieve high scalability in a cost-effective manner. In P2P systems, the overall system load is distributed among all participating users/nodes, which organize themselves into an overlay network in which they all have an equal role and act as both clients and servers.

A key aspect of P2P networks is the overlay topology structure that defines the way peers connect to other "neighboring" peers with whom they can interact and exchange messages. The overlay topology can be arbitrary or, alternatively, can be inspired by some application-specific semantics. In a typical NVE, a user sees only a portion of the virtual world where she/he can perform actions (i.e., moving around, manipulating objects, communicating with other users, etc.). This portion of the virtual world, commonly known as Area of Interest (AOI), is often based on virtual proximity (i.e., distance in the virtual world) between users. A user is thus only interested in the activities happening within its AOI. In this context, it is essential to dynamically organize the overlay network with respect to users' positions in the virtual world by having each user only connect to the set of neighboring users lying in her/his vicinity, and ensure that a user only receives state update messages of events happening within its AOI.

Towards this end, the well-known *Delaunay Triangulation* (DT) [15] in computational geometry can be used to provide connectivity between NVE users based on virtual proximity. The Delaunay Triangulation provides nice features that make it particularly attractive for P2P-based NVEs. In particular, it provides locality by connecting every node the set of neighbors lying in her/his vicinity in the virtual world, thus enabling direct message exchange between nodes that are susceptible to interact. This locality feature also achieves scalability by limiting the number of neighbors that a node is connected to, independently from the size of the network. A global behavior is then achieved through cooperative local interactions.

The basic geometric property underlying the broad success of Delaunay Triangulation is that it provides the most overall balanced triangulation by avoiding long thin triangles with very acute internal angles. This feature is very appealing for a wide range of applications. However, using a Delaunay Triangulation in a highly dynamic environment, and particularly in the context of P2P-NVEs, clearly suffers from high maintenance cost as it is subject to high connection change rate (known as *edge-flip* operation) due to continuous users' movement [15]. As users tend to move in the virtual world, the network topology should continuously adapt to users' movements such that the triangulation is kept valid. Maintenance cost can become particularly high when crowding occurs in parts of the virtual world, and a highly dynamic interaction with the surrounding takes place.

In the light of the above observation, the following question naturally arises: *is Delaunay Triangulation necessary or simply sufficient for maintaining NVEs connectivity?*

In this paper, we argue that Delaunay Triangulation is very restrictive in the context of NVEs. More precisely, we propose a new triangulation algorithm by slightly relaxing DT's equiangular property constraint, while still maintaining a suitable overlay for P2P NVEs. We achieve our goal by maximizing the region where a user can freely move without generating a flip operation, therefore reducing the message cost for maintaining a valid overlay. Intuitively, one would expect that this be obtained at the cost of a higher latency/number of hops required for message delivery. Performance evaluations under various scenarios show that overall, our triangulation algorithm does not incur higher latency that DT, while drastically reducing overlay maintenance cost.

Interestingly enough, while our solution induces a slightly higher latency in a very particular scenario case, it outperforms the classical Delaunay Triangulation in all other scenarios as shown by our performance evaluations.

The important contribution of this paper is beyond the good performances of our proposed algorithm. It actually questions the broad consensus on using Delaunay Triangulation for maintaining connectivity in highly dynamic P2P NVEs, and lays down the basis for further investigations for alternative solutions specifically tailored for P2P NVEs.

The rest of the paper is organized as follows. Section 2 provides a description of existing related works. Section 3 recalls the well-known Delaunay Triangulation construct. Section 4 discusses our triangulation algorithm. Extensive performance evaluations are given in section 5. Section 6 provides a discussion of our solution, while section 7 concludes this work and points outs some of our future perspectives.

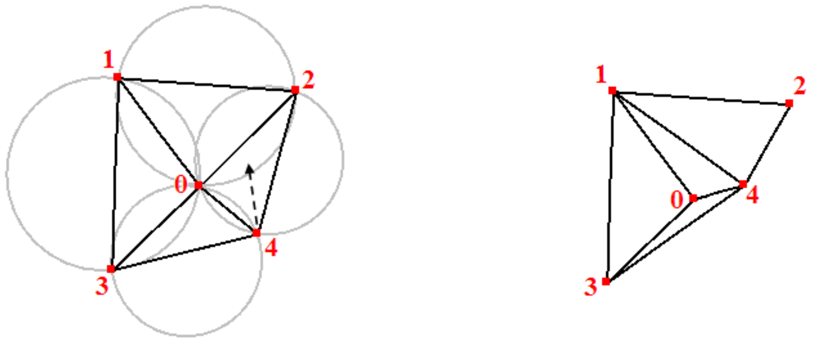
2 Related Works

A number of P2P designs have been recently proposed for scalable NVE support. A key issue common to all P2P-NVEs is dynamic topology maintenance. This implies that the network should cope with user movement through neighbor discovery and self-reorganization such that the overlay topology is kept consistent.

In [1], the authors propose a distributed NVE architecture based on the *Pastry* distributed hash table [3] and its corresponding application-layer multicast, *Scribe* [4]. The game world is partitioned into fixed-size regions, each managed by a unique *coordinator* node. The coordinator node of a region also serves as the root of a multicast tree to which all the peers lying in the region subscribe. State changes occurring in a region are subsequently multicast by the root node to all the region members. Coordinator nodes are randomly chosen and connected to each other using Pastry. [2] follows the same approach except that nodes of a region directly connect to the coordinator node of their region, thus eliminating message relay through direct connections. However, both works suffer from performance, scalability, and robustness issues inherent to server-based architectures as the coordinator can quickly become a bottleneck when crowding occurs.

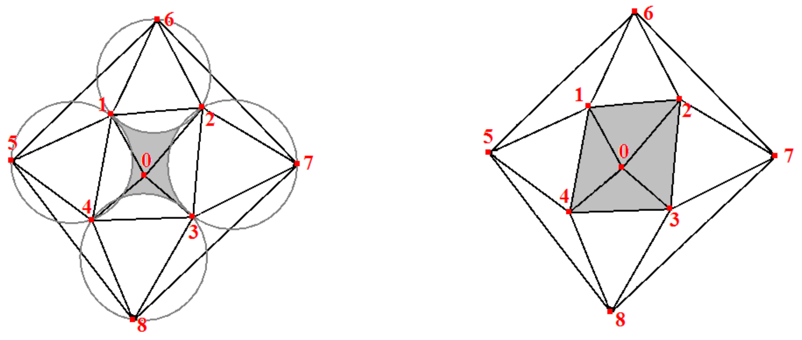
[7] describes fully-distributed P2P architecture for NVEs. Each node maintains a fixed number of direct connections to its closest neighboring nodes. Neighbor discovery is however achieved through regular neighbor-list exchange, which clearly introduces a high message overhead. [8] is a combination of DHT, unstructured overlay, and neighbor-list exchange. The virtual world is partitioned into hexagonal cells, each of which is assigned a unique *master* node present within the cell. A DHT is used to maintain the hierarchical structure and assign masters to cells. A cell's master keeps track of all other *slave* nodes in its cell, and regularly exchanges this list with master nodes of the neighboring cells. Slave nodes are notified of new neighbors by their master, while other message exchange is performed directly between the slaves. This scheme might however overload master nodes, and although neighbor-list exchange is performed only between masters (compared to [7]), its frequency is still a concern. [5, 6] is another fully-distributed architecture where each node maintains direct connections to all its AOI neighbors (i.e., all the nodes lying in its AOI). In case of position change, new neighbors are discovered through mutual cooperation and notification among direct neighbors. However, the proposed solution might lead to inconsistencies where some neighboring nodes go undetected.

The work presented in [9, 10] discusses a Voronoi-based partitioning of space in the context of networked virtual environments. Each node maintains a Voronoi diagram of its AOI neighbors, and keeps a direct connection to all of them. Neighbor discovery is again achieved by neighbor cooperation and mutual notification. [12] and [16] are extensions of [9] that add support to data and state



(a) Before **(b)** After

Fig. 1 (a) Delaunay Triangulation **(b)** Flip operation resulting from node *4*'s movement

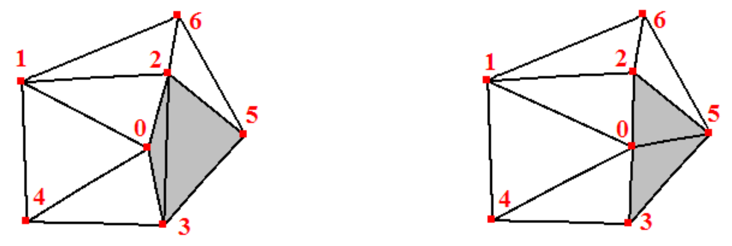


(a) **(b)**

Fig. 2 Node *0*'s flip-free area in **(a)** Delaunay **(b)** Triangulation algorithm

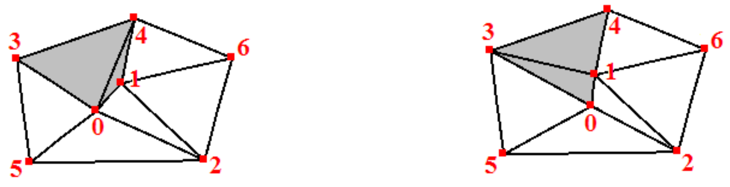
management in a fully distributed way. If crowding occurs, however, mechanisms that connect a node to all its AOI neighbors might lead to situations where almost all nodes become connected to each other due to overlapping AOIs (neighbor list size in $O(N^2)$, where N is the number of nodes in the system). This leads to obvious scalability problems where computing and bandwidth requirements exceed nodes' capabilities. To deal with this issue, [13] builds on the work presented in [9] to support AOI-Scalability. The work proposes to build a spanning tree across all AOI neighbors of a node, thus reducing bandwidth usage at the cost of a slightly higher latency.

However, none of the previously discussed works effectively deals with the high *maintenance* cost associated with the continuous dynamic organization of the overlay. This cost can become extremely high when crowding occurs, especially when it is associated with highly dynamic interactions between the nodes. As expected, these situations can be very frequent in NVEs in general and MMOGs in particular. Given this fact, [14] is, to the best of our knowledge, the first proposal to deal with the maintenance cost introduced by dynamicity and crowding. The authors describe a clustering scheme in a Delaunay-based overlay network. Every node in the network monitors its maintenance cost, and triggers a cluster creation procedure whenever its cost exceeds a given threshold. A cluster is then considered as a single node for the rest of the graph. Members of the same cluster then expand their coordinates by stretching their inter-node links. Apart from the cost associated with cluster creation and maintenance, the proposed mechanism indeed reduces the effect of density, but still performs all edge-flip operations dictated by the Delaunay Triangulation. This is exactly the issue tackled by our work. In this respect, our work can be regarded as orthogonal and both mechanisms can be combined, each bringing its own optimization.



(a) Before **(b)** After

Fig. 3 Flip operation resulting from crossing the base of the triangle during node *0*’s movement towards node *5*



(a) Before **(b)** After

Fig. 4 Flip operation resulting from crossing the base line of the triangle during node *0*’s movement towards node *2*

3 Delaunay Triangulation

The Delaunay Triangulation for a set of vertices A in a 2-D plane is a triangulation DT(A) such that no vertex of A is inside the circumcircle of any triangle in DT(A) (see Figure 1a) [15]. Note that a position change of a vertex in A can violate the basic property of DT. Let *0*, *2*, *1*, *4* be vertices in A and *021* and *042* be triangles in DT(A). If *4* changes position and enters inside the circumcircle of *021*, triangles *021* and *042* violate the Delaunay property. To meet the Delaunay condition, an *edge flip* operation is performed by switching the common edge *20* for the common edge *41* resulting in two valid triangles *041* and *214* (see Figure 1). The gray area in Figure 2a shows the area where node *0* is allowed to move without triggering any flip operation.

A Delaunay Triangulation thus provides a connected graph of a set of vertices based on their proximity. Furthermore, the average number of edges per vertex is small and independent of the size of A (generally less than six). For this reason, the Delaunay Triangulation constitutes an interesting choice for the structuring of P2P-NVEs. For our purposes, every user (or *node*) in the virtual world is represented by a vertex in the Delaunay graph, with user virtual positions (i.e., position in the virtual world) used as the corresponding vertex coordinates. Two nodes are direct neighbors (i.e., one-hop neighbors) in the overlay if their corresponding vertices in the Delaunay graph are connected via an edge. In the following, the terms "user", "node", and "vertex" will be used interchangeably. To maintain a valid topology in spite of node movements, every node has to inform its direct neighbors of its position change. This allows discovery of new neighbors through mutual notification where nodes collaborate to inform each others of new "approaching" nodes, and thus enables dynamic organization of the overlay through flip operations.

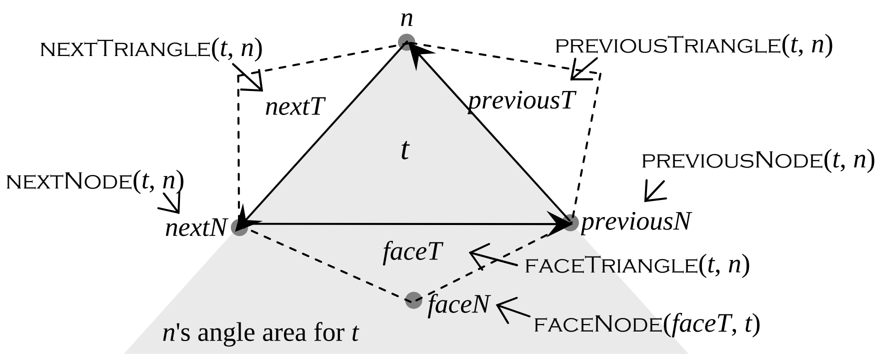


Fig. 5 A triangle *t* of *n*

4 Delaunay Triangulation Revisited

As stated earlier, a position change in Delaunay Triangulation might initiate a flip operation so that validity is maintained. NVEs are extremely dynamic environments due to continuous user movements. In order to minimize the maintenance cost resulting from high connection change rate due to user movements, we want to maximize the area where a node is allowed to move without triggering any flip operation (hereafter, we call this area a *flip-free* area). To this end, we modify the flip-free area of a node *N* to become the total of the triangular regions around *N*, i.e., the region composed of all the triangles for which *N* is a vertex and a flip operation occurs only when *N* passes across the base line of one of its triangles (see Figure 2b).

4.1 Node Movement

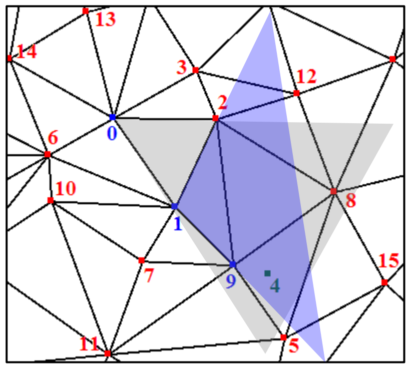
In our approach, during the movement of a node *N* inside the area containing all its triangular regions, a flip operation occurs only when *N* passes across the base line of one of its triangles. In Figure 3, when node *0* moves towards node *5*, node *0* passes across the base of triangle *032* and results in flipping the edge *32* to the edge *05*. As a consequence, the movement of node *0* deletes triangles *032* and *523*, and creates triangles *035* and *052*.

In Figure 4, when node *0* moves towards node *2*, node 0 passes across the base line *14* of triangle *014*. As a consequence, the movement of node *0* causes a flip operation between triangles *014* and *043* and creates triangles *013* and *431*. This kind of flip operations occurs when the total triangular region around a node *N* is not convex. As a result of *N*'s movement, a base of any triangle violating the convexity property of the total triangular region ends up with a flip operation switching this edge (base of triangle) with an edge that meets the convexity property. Apart from flip operations occurring while crossing the base line of a triangle, all other flip operations that might be triggered during a node movement in classical Delaunay Triangulation are eliminated.

The algorithm CONTROLTRIANGLEBASELINE checks if a moving node *n* passes across the base line of any of its triangles. If so, a flip operation occurs between the couple of related triangles. Figure 5 shows one of *n*'s triangles, say *t*, to illustrate the functions used in the algorithm for this triangle (i.e., the gray area indicates *n*'s angle area for *t*, NEXTTRIANGLE(*t*, *n*) returns the triangle *nextT*, NEXTNODE(*t*, *n*) returns the node *nextN*, and so on). Besides, GETFIRSTTRIANGLE(*n*) returns the first triangle of *n*'s triangle list and GETTRIANGLENUMBER(*n*) returns the size of *n*'s triangle list.

Algorithm 1 ControlTriangleBaseLine(*n*)

```
Input: n : a node which changes position in the virtual world
    t ← getFirstTriangle(n)
    index ← 0
    while index < getTriangleNumber(n)
        previousT ← previousTriangle(t, n)
        nextT ← nextTriangle(t, n)
        faceT ← faceTriangle(t, n)
        previousN ← previousNode(t, n)
        nextN ← nextNode(t, n)
        faceN ← faceNode(faceT, t)
        previousN2 ← previousNode(previousT, n)
        nextN2 ← nextNode(nextT, n)
        if p does not pass across the base line of t then
            { check if there is a flip between t and nextT }
            if p passes across the base line of nextT
              and nextN2 is inside previousN's angle area for t then
                Flip operation occurs between nextT and t
                t ← getFirstTriangle(n)
                index ← 0
            { check if there is a flip between t and previousT }
            else if p passes across the base line of previousT
              and previousN2 is inside nextN's angle area for t then
                Flip operation occurs between previousT and t
                t ← getFirstTriangle(n)
                index ← 0
            { switch to the next triangle in n's triangle list }
            else
                index ← index + 1
                t ← previousT
            endif
        { check if there is a flip between t and faceT }
        else if p is inside faceN's angle area for faceT then
            Flip operation occurs between t and faceT
            t ← getFirstTriangle(n)
            index ← 0
        { switch to the next triangle in n's triangle list }
        else
            index ← index + 1
            t ← previousT
        endif
    endwhile
end
```



▲ *0*'s angle area for triangle *012*

▲ *1*'s angle area for triangle *192*

🟩 New node

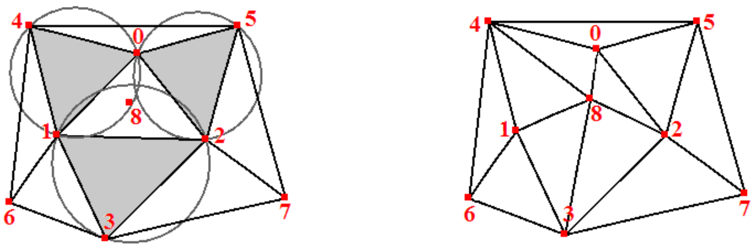
Fig. 6 Transmission of node *4* join request

```
Algorithm 2 INSERTION(n, p)
Input: n : a node in the virtual world, p : node wants to enter the virtual world
    t ← GETFIRSTTRIANGLE(n)
    index ← 0
    while index < GETTRIANGLENUMBER(n)
        if p is inside t then
            n inserts p in the world
        else if p is inside n's angle area for t then
            previousN ← PREVIOUSNODE(t, n)
            nextN ← NEXTNODE(t, n)
            closestN ← FINDCLOSESTNODE(nextN, previousN, p)
            INSERTION(closestN, p)
        else
            index ← index + 1
            t ← PREVIOUSTRIANGLE(t, n)
        endif
    endwhile
end
```

4.2 Node insertion

For the node insertion algorithm, we define the *angle area* of a node *N* for a triangle *T* as the extended area of the angle of *T* having *N* as a vertex (see Figure 6). A node *P* that wants to enter to system first contacts a gate node[1], say *N*. Gate node *N* first checks if *P* is inside one of its triangles. If so, *N* allows *P* to enter the system according to the insertion algorithm. If not, *N* transmits *P*'s entry demand to *N*'s closest neighbor to *P*, say *K*, selected from the triangle for which *N*'s angle area contains *P*. *K* performs in turn the same operation. This operation continues until the entry message reaches the node allowing node *A* to enter the system. Figure 6 shows the transmission path of node *4*'s entry demand beginning from the gate node *0* up to node *9*. Since node *4* is inside node *0*'s angle area for triangle *012*, node 0 transmits *4*'s entry demand to node *1*, the closest among *1* and *2* to *4* in the virtual world. Similarly, node *1* transmits *4*'s demand to node *9*, which allows node *4* to enter to the system. The algorithm INSERTION allows the new node *p* to enter the system. The procedure FINDCLOSESTNODE(*n1*, *n2*, *p*) returns the closest node to node *p* between nodes *n1* and *n2*.

1 We assume that there is a set of nodes, called gate nodes, which IP addresses are known by the system. Gate nodes allow new nodes to join the virtual world.



(a) Before **(b)** After

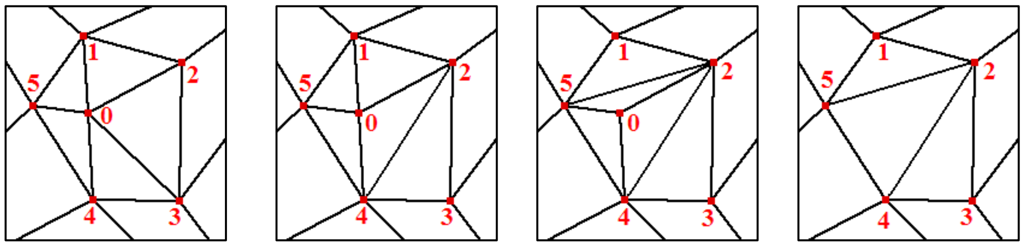
Fig. 7 Insertion of node *8* resulting in two flip operations with neighbor triangles

Let *T* be the triangle inside which the new node *P* enters. *T* has three neighbor triangles with which *T* has a common edge. When *P* enters inside *T*, we also check whether *P* enters inside the circumcircle of *T*'s neighbor triangles. If so, a flip operation is performed with the neighbor triangle inside the circumcircle of which *P* enters. Figure 7 shows the insertion of node *8*. Let node *0* be the node allowing node *8* to enter the system. Node *0* finds that node *8* enters inside triangle *012*. Node *0* also controls if node *8* enters inside the circumcircle of neighbor triangles *041* and *025* and so a flip operation is performed between triangles *012* and *041*. The control with the third neighbor triangle *123* is done by one of *0*'s neighbors (either node *1* or node *2*) of triangle *012*. Then the second flip operation is performed between triangles *012* and *321*. The reason behind limiting the control operation to the circumcircles of the three neighbor triangles is to avoid successive flip operations in the virtual world (as would have been generated by standard Delaunay). As an example, in Figure 7, node *8* can also be inside the circumcircle of triangles *163*, *146* and *237*, meaning that several flip operations can be performed causing several message transmissions between the nodes.

4.3 Node deletion

A node *N* that wants to quit the system first starts by decreasing the number of its triangles. To do so, successive flip operations occur between *N*'s triangles until there is no possible flip operation between any of its triangles. At this point, two cases are possible: (1) *N* has three remaining neighbors, or (2) *N* has four remaining neighbors. In the following, we examine node deletion in detail, considering both cases.

In Figure 8a, node *0* wants to quit the system. Triangles *043* and *032* are two adjacent triangles of *0*. Since node *2* is in the angle area of node *4* for triangle *043*, the first flip operation occurs between triangles *043* and *032*, and are replaced by triangles *042* and *324* (see Figure 8b). Figure 8c shows *0*'s triangles after the second flip operation occurring between triangles *021* and *015*. Since there is no more possible flip operations between *0*'s triangles (Figure 8c), *0*'s three remaining neighbors form a new triangle *254* (Figure 8d). Node *0* deletes all its triangles and quits the world. If, on the other hand, four neighbors remain for node *0* after performing the successive flip operations among *0*'s triangles (which is the only other possible case), *0*'s four neighbors form two triangles (Figure 9). Similarly, node *0* deletes all its triangles and quits the world. The algorithm DELETION gives the details of the deletion of a node.



(a) **(b)** **(c)** **(d)**

Fig. 8 Deletion of node *0*

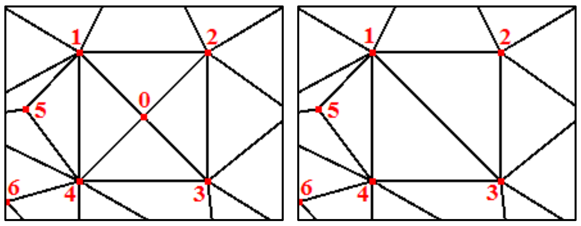


Fig. 9 Deletion of node *0* having four neighbors

Algorithm 3 DELETION(*n*)

```
Input: n : a node in the virtual world
    t ← GETFIRSTTRIANGLE(n)
    index ← 0
    { All possible flip operations are performed between n's triangle }
    while index < GETTRIANGLENUMBER(n)
        previousT ← PREVIOUSTRIANGLE(t, n)
        nextN ← NEXTNODE(t, n)
        previousN2 ← PREVIOUSNODE(previousT, n)
        if previousN2 is inside nextN's angle area within t then
            Flip operation is performed between previousT and t
            t ← n's new triangle resulted from the flip operation
        else
            index ← index + 1
            t ← previousT
        endif
    endwhile
    { n is a node inside a triangle }
    if GETTRIANGLENUMBER(n) is 3 then
        n's 3 triangle neighbors form a new triangle
    else { n is connected to 4 nodes }
        n's 4 triangle neighbors form two new triangles
    endif
    n removes its triangles
    n quits the world
end
```

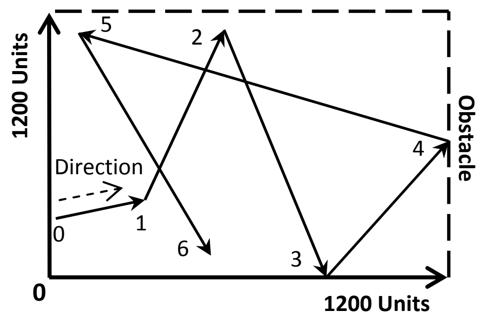
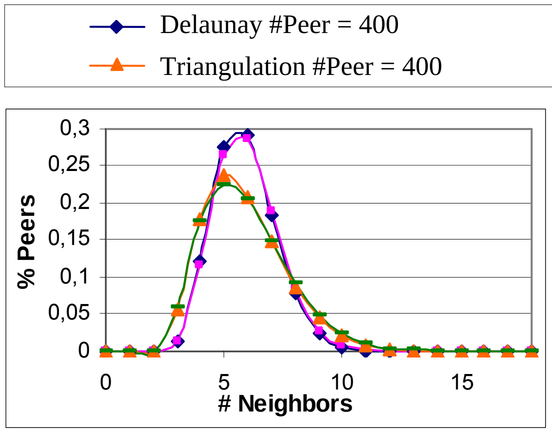


Fig. 10 Random movement of a peer



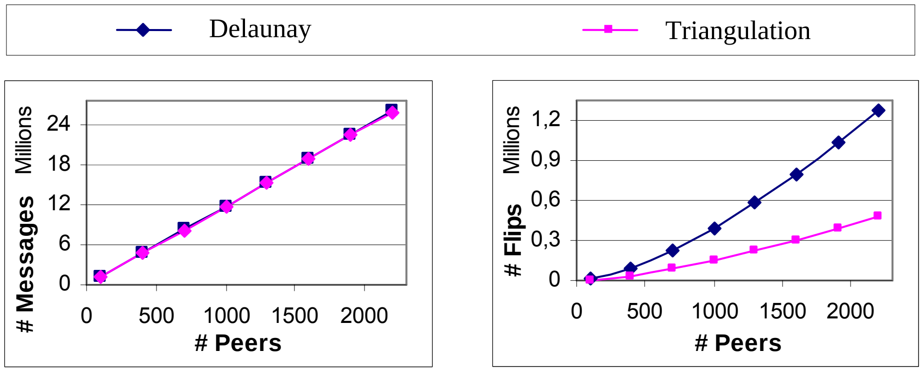
Delaunay		
# Peer	**Average**	**Standard Deviation**
400	5.861	1.304
1300	5.898	1.339

Triangulation		
# Peer	**Average**	**Standard Deviation**
400	5.831	1.760
1300	5.868	1.824

Fig. 11 Percentage of peers having *X* triangle neighbors

5 Experimental Evaluations

In this section, we present the results of our simulation. We compare our Triangulation algorithm with the classical Delaunay algorithm under four different movement scenarios. Notice that the movement of a peer concerns two types of actions: (1) informing triangle neighbors about position change, and (2) performing possible flip operations. Note that the communication cost of a flip operation is higher than a simple sending of a position change message to a neighbor. This is due to the fact that a flip operation involves message exchange between at least three peers, where two peers that do not have information about the existence of each other become connected by the help of a third peer. To illustrate, consider again Figure 3. The flip operation resulting from node *0*'s movement towards node *5* occurs in the following way: while node *0* is moving, it regularly sends a position update to its neighbors, namely nodes *1*, *2*, *3* and *4*. When node *2* (or node *3*) discovers that node *0* has reached/crossed base line *23*, it informs node *0* that a flip operation should occur by sending it a message containing the identity of node *5* to which node *0* should connect. Consequently, node *0* sends a connection request message to node *5*, which includes all the necessary information for node *5* about node *0* (physical and virtual addresses, port number, etc.). Upon receipt of the connection request, node *5* sends node *0* an (application-layer) acknowledgement message including all the needed information about node *5*. This establishes a new edge between node 0 and node 5. Consequently, the flip operation would have resulted in three message exchanges, compared to a simple sending of a position update. Given this communication cost difference, each of these parameters is evaluated separately.



(a) Number of messages to triangle neighbors **(b)** Number of flip operations

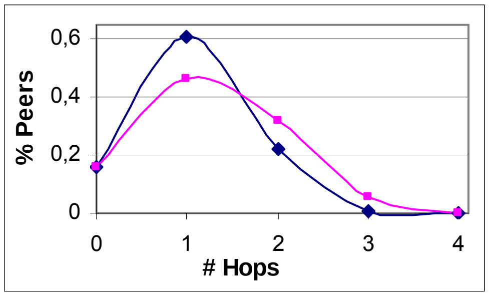
Fig. 12 Random movement

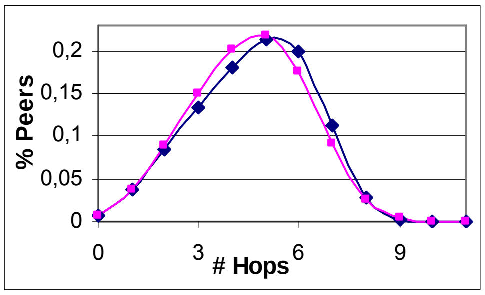
5.1 Random movement

In the random movement scenario, peers move randomly in the world. Each peer moves towards a given direction during a given period of time. If the peer meets any obstacle on its path or if the time period expires, the direction and the period are reinitialized and the peer continues to move in its new direction. We consider a two dimensional virtual world of 1200x1200 units. The simulations proceed in over 2000 discrete time-steps. The evaluation is performed with a number of peers varying between 100 and 2200 in 300 increments. Figure 10 shows the example of the random movement of a peer changing 5 times its direction. Figure 11 represents the percentage of peers having *X* neighbors. We use the evaluation result of a system with 400 and 1300 peers connected based on Delaunay algorithm and our triangulation algorithm (henceforth referred to as Triangulation).

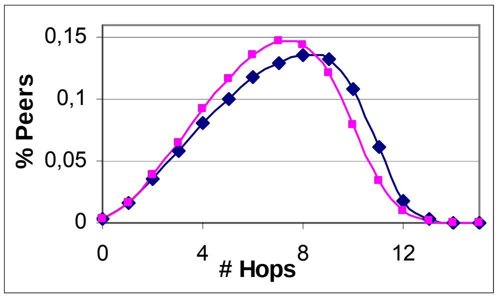
The average number of triangle neighbors is approximately 6 for both algorithms. Note that the standard variation of this average in Delaunay is lower than the standard variation in our Triangulation. This means that in Delaunay, the number of neighbors remains closer to the average number than in Triangulation. While it might seem advantageous, this characteristic of Delaunay however forces peers to have a strict number of neighbors that they are not allowed to change, which is not the case in Triangulation. During our evaluation of Triangulation, a flip operation is *only* performed when a peer passes across the base line of one of its triangles. However, Triangulation allows also a flip operation to be performed even if no crossing of a triangle's base line occurs (for instance, when the peer decides to drop the farthest neighbor). This flexibility allows peers to determine/adapt the number of neighbors to which they can connect. This property can be very valuable in real systems where peers are heterogeneous in terms of performance, capacity, energy, etc. Therefore, Triangulation algorithm allows peers to increase/decrease the number of their neighbors with respect to both their own conditions/requirements and the system's ones. It is also important to note that the Triangulation topology becomes more similar to Delaunay topology as the number of neighbors approaches the average.

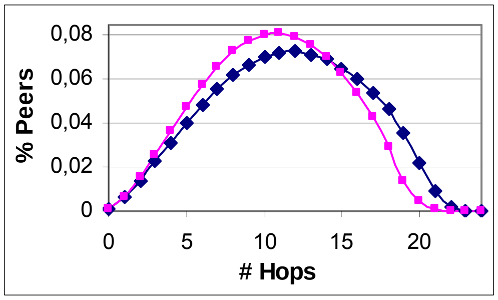
Figure 12 illustrates the comparison of Delaunay and Triangulation algorithms for the random movement scenario. Figure 12a shows the total number of position update messages sent by each peer to its triangle neighbors. The number of position update messages is the same for Delaunay and Triangulation algorithms. This is clearly due to the fact that even though the standard deviation in the number of neighbors is different in both algorithms (as seen in Figure 11), the average number of neighbors is very close; therefore, we obtain similar curves. However, in Figure 12b, we clearly see the performance difference between the two algorithms in terms of the number of the flip operations performed during peers' movements. When the number of peers increases, the number of flip operations increases more rapidly in the system based on Delaunay algorithm. More precisely, the number of flip operations performed in a Delaunay-based system is 2.65 times more than a Triangulation-based system.





(a) radius = 50 **(b)** radius = 300





(c) radius = 500 **(d)** radius = 1100

	Average # Hops		**Standard Deviation**	
r AOI	**Delaunay**	**Triangulation**	**Delaunay**	**Triangulation**
50	1.077	1.275	0.644	0.807
300	4.637	4.506	1.743	1.719
500	6.997	6.610	2.634	2.475
1100	11.546	10.617	4.694	4.259

(e) Average and standard deviation values for Delaunay and Triangulation algorithms

`peerID`	`messageID`	`counter`	`radiusAOI`	`X`	`Y`	`senderID`

(f) Structure of a position update message

Fig. 13 Percentage of peers receiving a message after *X* hops

In Figure 13, we compare message broadcast cost for Triangulation and Delaunay topologies. To do so, we assume that each position update message is broadcast not only to its triangle neighbors but also to all other peers lying inside the peer's *area of interest* AOI (i.e., the peers for whom the movement is visible). For simplicity, we define a peer's AOI by a circle whose center point coincides with the position of the moving peer.

Figure 13f illustrates the structure of a position update message. *PeerID* is the *id* of the moving peer. *MessageID* is the id of the message. *Counter* helps with determining the number of previous hops the message has made before reaching a peer. *RadiusAOI* is the AOI radius of the moving peer. *X* and *Y* are the coordinates of the moving peer. *SenderID* is the *id* of the sender of the message, which helps with avoiding sending the same message back to its previous sender. Each peer keeps the last relevant received message from a peer.

When a peer changes its position in the world, it sends to each of its triangle neighbors a message containing its *id*, the *id* of the last sent message augmented by one, the counter value initialized to 0, the radius of its AOI, its coordinates and again its *id* as sender. When a peer *P* receives a message *M* [*pID* | *mID* | *c* | *r* | *X* | *Y* | *sID*], one of the following occurs :

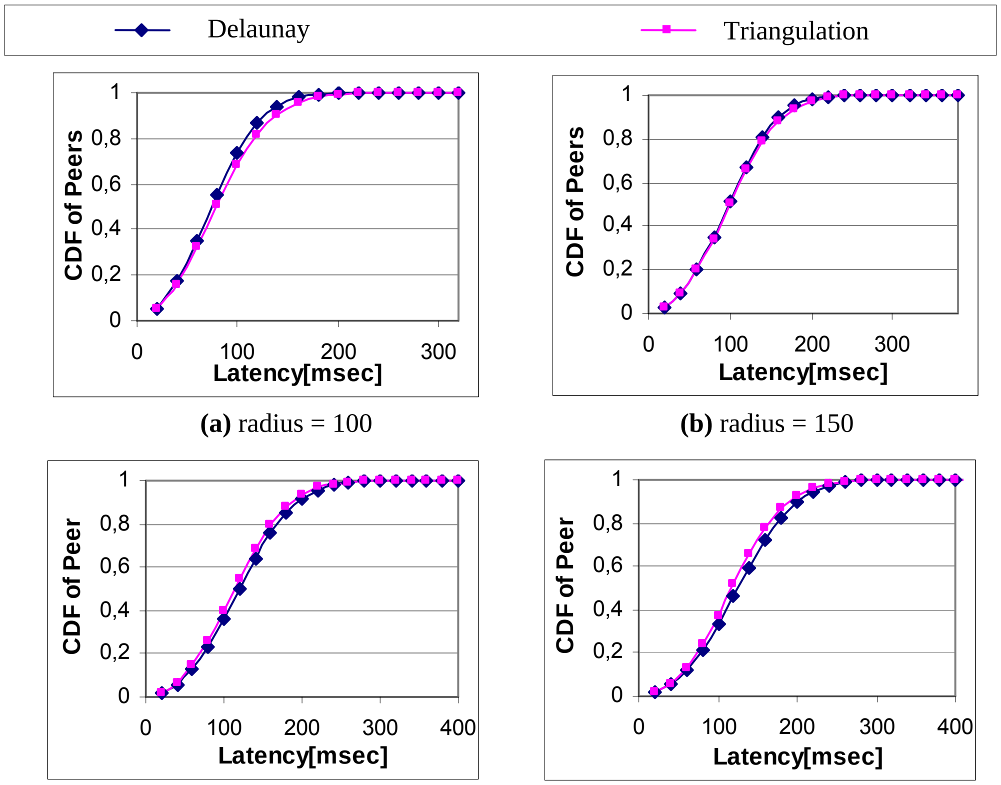
- If *P* does not have any message with *pID*, *P* keeps *M* and transmits the message by incrementing *c* by one, to its triangle neighbors (except to the peer sID) which are inside the AOI of the peer *pID*.
- If *P* has a message *M2* [*pID* | *mID2* | …], and
  - If *mID2* < *mID*, *P* replaces *M2* par *M*, and transmits the message, by incrementing c by one, to its triangle neighbors (except to the peer sID).
  - If *mID2* ≥ *mID, P* discards *M*.

In this evaluation, there are 1000 peers moving randomly in the world of 1200x1200 units during 2000 time-steps. Figure 13 shows the percentage of peers, inside the moving peer's circle with radius *r*, receiving a message after *X* hops from its originator. Figure 13a shows the evaluation results when the circle radius is equal to 50 units, meaning that each time a peer changes position in the world, all peers closer than 50 units to the moving peer receive its position updates.

Figure 13d shows the situation where nearly all peers in the world are informed about each position change. We see that when the value of the radius is low, meaning that an update message is only sent to the peers nearby the moving peer, Delaunay overlay provides better performances in terms of the number of logical hops than Triangulation. This was to be expected given that in Triangulation, the next hop routing choice is no longer guaranteed to be the closest neighbor to the destination node in the virtual world. However, the situation is reversed and Triangulation provides better performances when the radius of the AOI starts to increase (i.e., peers also want to communicate position updates to distant peers in the world). This is due to the fact that in Triangulation, the probability for connecting peers that are far away from each other is higher than in Delaunay (where locality guarantees are stronger). Similarly to long range links in augmented graphs models, these connections play the role of shortcuts that enable messages to reach their destination much faster than in a more conservative topology.

As shown in Figure 13e, the algorithm with the lowest average number of hops has also the lowest standard deviation value, meaning that the difference between the minimum and the maximum number of hops of the same message received by the peers inside the circle is lower, and most of the peers inside the circle receive the message after the average number of hops.

However, the construction of the overlay network based on virtual positions of users in the virtual world causes topology mismatch between the logical overlay and the underlying physical network. This means that the number of logical communication hops does not reflect the latency of the physical network. Therefore, an increased number of hops does not necessarily imply a higher latency (and vice versa). In order to study the latency of both algorithms under various scenarios, we analyze the network delay effect on communication cost. To do so, we attribute randomly selected network latencies with values between 5 and 100 *msec*. We particularly evaluate the case when the



(a) radius = 100 **(b)** radius = 150

(c) radius = 200 **(d)** radius = 250

Fig. 14 Cumulative distribution function of peers receiving a message before *X msec*

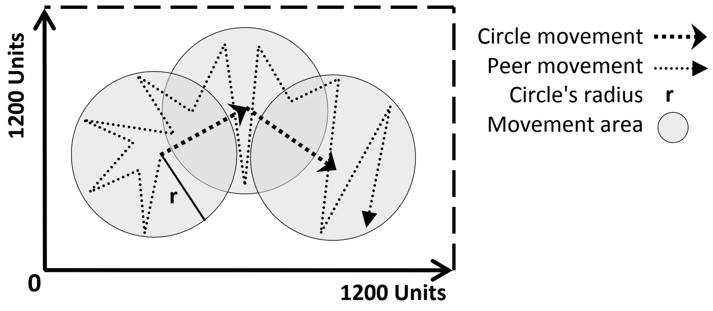
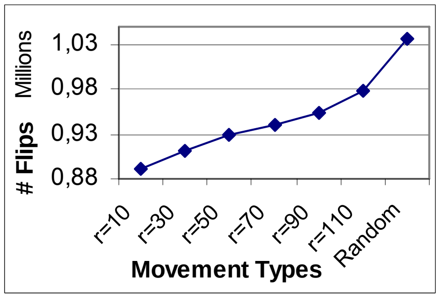
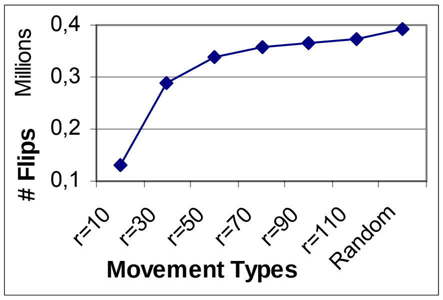


Fig. 15 Restricted movement of a peer

value of the AOI radius is small, for which Delaunay provides better performances than Triangulation in terms of the number of logical communication hops. Figure 14 shows the cumulative distribution function of peers receiving a message before *X msec* when the value of the AOI radius is between 100 and 250 in 50 increments. The performance of Delaunay is slightly better when the radius value is 100 (Figure 14a), nearly equal when the radius is 150 (Figure 14b) and Triangulation provides better performances when the radius value ≥ 200 (Figure 14c, 14d). Starting at 200, the larger the AOI radius gets, the more the Triangulation algorithm outperforms Delaunay.



(a) Delaunay algorithm (#Peer = 1900)

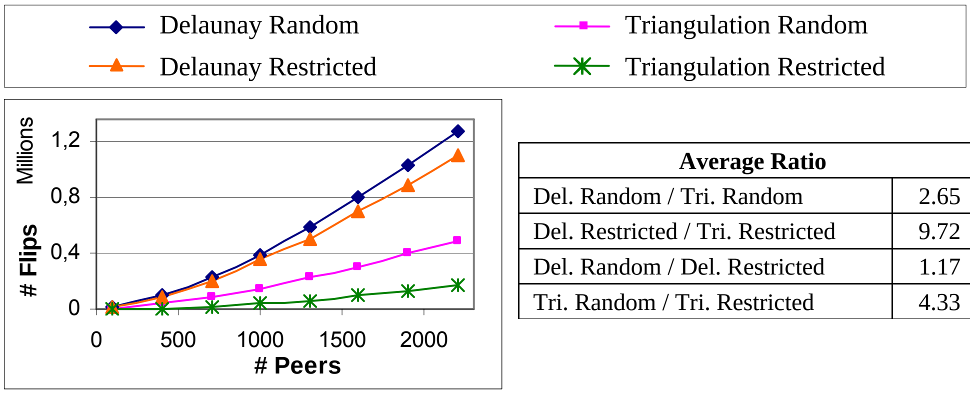


(b) Triangulation algorithm (#Peer = 1900)

Radius (r)	10	30	50	70	90	110
#Flip Delaunay / #Flip Triangulation	6.88	3.15	2.75	2.64	2.60	2.61

(c) Comparison of number of flip operations in Delaunay & Triangulation algorithms

Fig. 16 Number of flip operations in Random and Restricted movements for different values of *r*



Average Ratio	
Del. Random / Tri. Random	2.65
Del. Restricted / Tri. Restricted	9.72
Del. Random / Del. Restricted	1.17
Tri. Random / Tri. Restricted	4.33

Fig. 17 Comparison of algorithms in Random & Restricted movement (*r* = 10)

5.2 Restricted movement

In the random movement scenario, peers move throughout the world. However, peers have generally a stronger *locality of interest* in the nearby regions around them, and consequently perform their movements inside their local environment. In such local movement scenarios, our algorithm has even much better performances compared to Delaunay triangulation. In a locally restricted movement scenario, a peer moves inside a circle centered on the initial position of the peer. With time, the center position of the circle also moves in the world. In other words, a peer's movement is limited by a circle around it, and this circle also slightly changes position in the world. Figure 15 shows the example of the restricted movement of a peer where the movement circle changes also position with time.

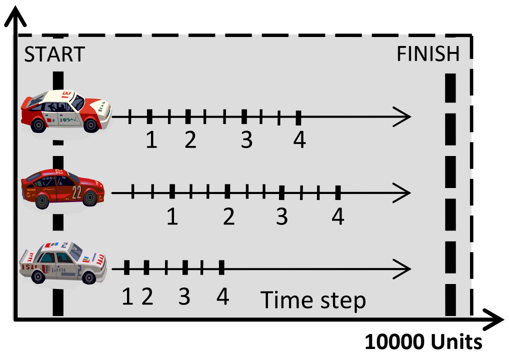


Fig. 18 Race game

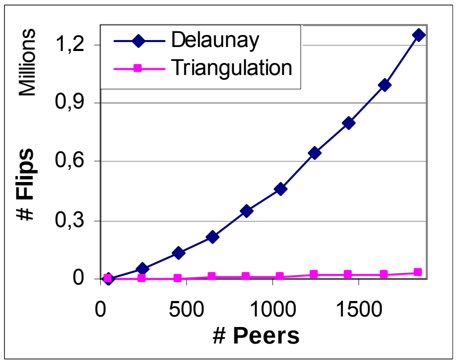
Figure 16 shows the number of flip operations for both Delaunay and Triangulation algorithms. The evaluation is performed in a world of 1200x1200 units with 1900 peers moving inside a circle with radius *r* taking values between 10 and 110 in 20 increments. When the radius decreases, the number of flip operations also decreases for both algorithms. However, the number of flip operations in Triangulation is extremely small for small radius restricted movements. Table 16c gives the ratio of the flip operations in Delaunay algorithm versus in Triangulation algorithm.

Figure 17 compares the two algorithms while incrementing the number of peers in the system from 100 to 2200 in both random and restricted movements when the radius of the restricted movement is equal to 10. We see that in random movement, the number of flip operations performed in Delaunay is 2.65 times more than Triangulation. However, in restricted movement, this increases up to 9.72 times more.

5.3 Race game

We also evaluate the performance of our Triangulation algorithm through a simple race game simulation. In the race game (illustrated in Figure 18), participants are aligned at the beginning at the "Start" line (i.e., they have the same X coordinate value but different Y coordinate values). Participants' goal is to reach the "Finish" line by moving horizontally on a 10000 unit-length linear race (i.e., during the game, the Y coordinates of participants do not change). Each participant has a speed, a duration value and a speed mode. The speed of a participant indicates how fast he runs. The duration value determines how long he runs with the same speed. The speed mode, which can be either positive or negative, determines if the participant speed is either in *increase* or *decrease* mode. If the speed mode is positive, the participant will run with a higher speed during the next duration. At the beginning of the game, the speed mode of all racers initializes to positive, the speed and the duration values are initialized to random values respectively between 1 and 6 units/time-steps and between 5 and 100 time-steps. At each time step, the X coordinates of participants change according to their speed and their duration value decreases by one. If the duration becomes zero, the duration is reinitialized to a random value and the racer's speed either increases or decreases by 1 depending on the racer's speed mode. A positive speed mode becomes negative when the racer's speed reaches the maximal speed, 6 unit/time-step. Similarly, a negative speed mode becomes positive when the racer's speed reaches the min speed, 1 unit/time-step.

Figure 19 shows the number of flip operations performed in both algorithms while incrementing the number of participants from 50 to 1850 in 200 increments. As shown in the table, for a race with 1850 participants, the number of flip operations performed in Delaunay algorithm is 44.91 times more than the Triangulation algorithm. The important performance difference between the two algorithms comes from the fact that in Triangulation, a triangle from a previous time-step stays valid most of the times in the next step since the three points of the triangle change positions towards the same direction (i.e., all participants move towards the same direction in order to be closer to the Finish line of the game). However, in Delaunay, a flip operation can also be performed because of the circumcircle property (i.e., no other point in the circumcircle of a triangle). Thus, a point that is close to a triangle may enter the circumcircle of the triangle in the next time-step, which generates a flip operation.



# Peer	Delaunay / Triangulation
50	13.57
250	21.72
450	25.76
650	29.76
850	33.44
1050	35.20
1250	38.48
1450	39.20
1650	42.41
1850	44.91

Fig. 19 Comparison of both algorithms in a Race game

5.4 Second Life

Second Life is a popular online virtual world that enables its user residents to explore a digital universe and interact via their digital representations called "avatar" [17]. Liang et al. [18] analyze the traces of user movement in Second Life to examine avatars mobility and behavior. We use the traces collected by Liang et al. to evaluate our Triangulation algorithm and to compare it with Delaunay. The virtual world in Second Life is made up of regions, each of which is 256*m* × 256*m*. The traces of avatars moving around within a region are collected during one-day time period. In our evaluations, we use the avatar traces collected from the Freebies region in the virtual world. Avatar movements are registered every 10 seconds during 24 hours. Compared to existing virtual world applications, a 10-second time interval is extremely large for informing nodes about position updates. Therefore, we divide a 10-second time interval into 500-*msec* intervals and evaluate nodes' movements for each 500 *msec* (i.e., Δt = 500 *msec*). In order to calculate the movement of a node (Δx, Δy) per Δt, we divide the distance between the node's position at time t_i and at time t_{i+1} (i.e., $t_{i+1} = t_i$ +10sec) by (10sec / Δt), as follows:

$$\Delta t = 500 \;\; m\sec, \;\; \Delta x = \frac{x(t_{i+1}) - x(t_i)}{10\sec / \Delta t}, \;\; \Delta y = \frac{y(t_{i+1}) - y(t_i)}{10\sec / \Delta t}$$

Figure 20 represents the cumulative distribution function of peers having less than *X* neighbors in both Delaunay and Triangulation algorithms. Figure 21a shows the total number of update messages sent by each peer to its triangle neighbors. The number of update messages in the Triangulation algorithm is slightly higher than that in Delaunay. This is due to the difference in the number of neighbors in both algorithms (Figure 20). In Figure 21b, we clearly see the performance difference between the two algorithms in terms of the number of the flip operations.

In Figure 22, we compare the latency of both algorithms with respect to the latency of the underlying physical network. We attribute randomly selected network link latencies with values between 5 and 100 *msec*. An update message of a peer is sent to all other peers lying inside the peer's AOI. The evaluation is performed for an AOI radius varying between 100 and 250 in 50 increments. Figure 22 shows that overall, the Triangulation algorithm does not incur higher latency than Delaunay.

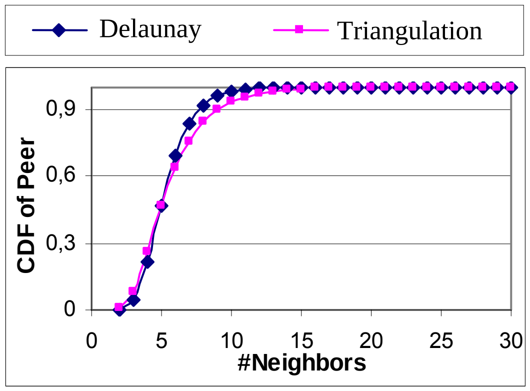
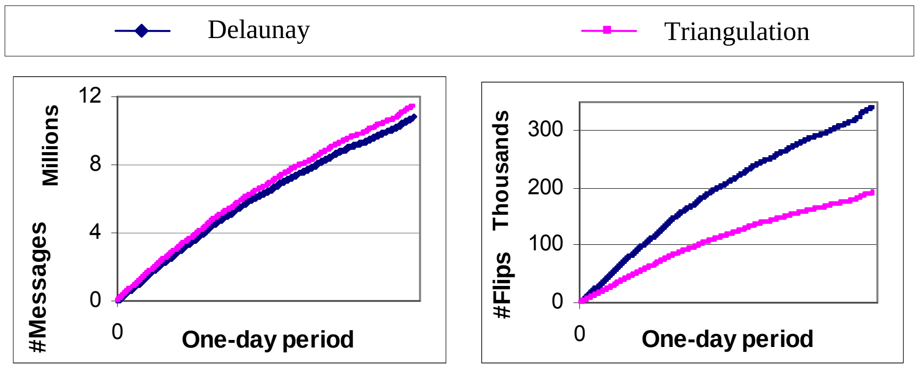


Fig. 20 Cumulative distribution function of peers having smaller than *X* triangle neighbors

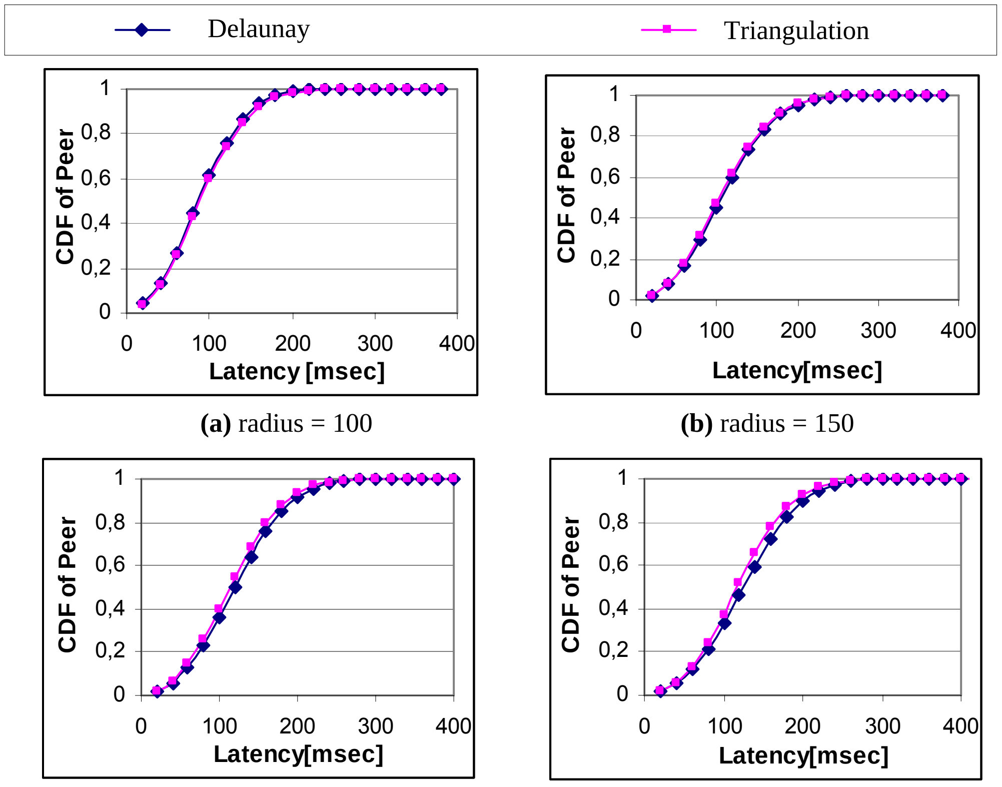


(a) Number of messages to triangle neighbors **(b)** Number of flip operations

Fig. 21 Second Life

5.5 Node insertion

Peers join the world at random positions. We evaluate the cost of peers' entrance procedure according to both algorithms. Figure 23 shows the number of flip operations generated by peers' entrance to the virtual world. In our algorithm, the join of a new peer affects at most four neighboring triangles. Therefore, our triangulation algorithm is more efficient than Delaunay in terms of the number of messages exchanged for peers' entrance in the system.



(a) radius = 100 **(b)** radius = 150

(c) radius = 200 **(d)** radius = 250

Fig. 22 Cumulative distribution function of peers receiving a message before *X* delays

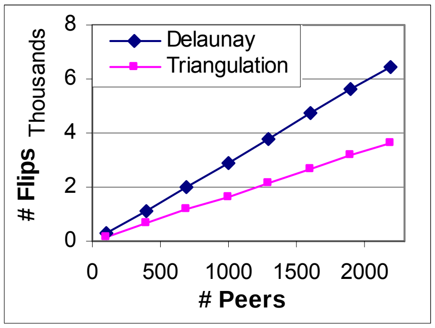


Fig. 23 Number of flip operations during a join procedure

6 Discussion

One of the main thoughts that come into one's mind is that Delaunay Triangulation allows distributed state management through the use of its corresponding dual structure, i.e., the Voronoi diagram [11].

A Voronoi diagram is mostly useful for distributed object state management, where the virtual world is partitioned into regions equal to the number of nodes such that each node lies in the region that contains all the points that are closer to it than to any other node. Simply stated, a Voronoi diagram defines the region that each node has to manage, while the Delaunay triangulation determines the neighbors to which the node is to be directly connected. Destroying the Delaunay structure clearly eliminates the possibility of using Voronoi for state management. Our thoughts on this issue can be resumed as follows:

While it appears natural that the virtual world decomposition among the peers, which translates their dynamically changing locality of interest, captures this evolution (as is case in Voronoi regions), it is not feasible from a performance perspective to couple object state management and ownership with the continuously evolving nature of the game world. Furthermore, such a scheme makes objects' ownerships very hard to determine (if not impossible), jeopardizing the system's performances and stability. Therefore, we believe that a good design choice would be to decouple object ownership for state management from users' positions in the virtual world (which eliminates Voronoi as an option). One possibility is to maintain two separate overlays/structures: a user overlay for connecting users in the game world depending on their virtual position and proximity, and a more stable structure for managing objects' ownerships and states.

7 Conclusion

Peer-to-peer (P2P) architectures have recently become a very popular design choice for building scalable Networked Virtual Environments (NVEs). While the well-known Delaunay Triangulation has several nice features that make it an interesting choice to provide connectivity between NVE users based on proximity, it introduces a high topology maintenance cost as it is subject to high connection change rate due to continuous user movement in the virtual world.

The contribution of this paper is twofold. First, we revisited the Delaunay Triangulation in the context of NVEs, and proposed a new triangulation algorithm that drastically reduces the overlay maintenance overhead. This is achieved by maximizing the region where a user can freely move without generating an edge-flip operation due to users' insertion and movement, therefore reducing the message cost for maintaining a valid overlay. Our mechanism proves particularly efficient in highly dynamic environments, and achieves even better performances when crowding occurs in parts of the virtual world and a highly dynamic interaction with the surrounding takes place. We have evaluated our mechanism through simulation and shown that it drastically reduces overlay maintenance cost in such situations, while even reducing the average number of message hops in a wide range of scenarios.

From the above emerges a more general contribution through which the broad consensus on using Delaunay for maintaining user connectivity in highly dynamic P2P NVEs is questioned. This opens the door for more research into alternative solutions designed with P2P NVEs requirements and constraints in mind.

9 References

[1] Knutsson B, Lu H, Xu W, Hopkins B (2004) Peer-to-peer support for massively multiplayer games. In: Proc. IEEE INFOCOM, pp 96-107

[2] Iimura T, Hazeyama H, Kadobayashi Y (2004) Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In: Proc. NetGames, pp 116-120

[3] Druschel P, Rowstron A (2001) Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proc. of Middleware, pp 329-350

[4] Castro M, Druschel P, Kermarrec A, Rowstron A (2002) SCRIBE: a large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications 20(8):1489–1499

[5] Keller J, Simon G (2003) Solipsis: a massively multi-participant virtual world. In: Proc. of PDPTA, pp 262-268

[6] Frey D, Royan J, Piegay R, Kermarrec AM, Anceaume E, Le Fessant F (2008) Solipsis: a decentralized architecture for virtual environments. In: Proc. MMVE, pp 29-33

[7] Kawahara Y, Morikawa H, Aoyama T (2002) A peer-to-peer message exchange scheme for large scale networked virtual environments. In: Proc. of IEEE ICCS, pp 957-961

[8] Yu A, Vuong ST (2005) MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In: Proc. of NOSSDAV, pp 99-104

[9] Hu SY, Liao GM (2004) Scalable peer-to-peer networked virtual environment. In: Proc. of NetGames, pp 129-133

[10] Hu SY, Chen JF, Chen TH (2006) VON: a scalable peer-to-peer network for virtual environments. IEEE Network 20(4):22–31

[11] Aurenhammer F (1991) Voronoi diagrams-a survey of a fundamental geometric data structure. ACM Computing Surveys 23(3):345-405

[12] Buyukkaya E, Abdallah M (2008) Data management in Voronoi-based P2P gaming. In: Proc. of IEEE CCNC Int. Workshop on Digital Entertainment, Networked Virtual Environments, and Creative Technology, pp 1050-1053

[13] Jiang JR, Huang YL, Hu SY (2008) Scalable AOI-cast for peer-to-peer networked virtual environments. In: Proc. of ICDCSW CDS

[14] Varvello M, Biersack E, Diot C (2007) Dynamic clustering in Delaunay-based P2P networked virtual environments. In: Proc. of NetGames, pp 105-110

[15] de Berg M, van Kreveld M, Overmars M, Schwarzkopf O (1997) Computational geometry, algorithms and applications. Springer-Verlag

[16] Hu SY, Chang SC, Jiang JR (2008) Voronoi state management for peer-to-peer massively multiplayer online games. In: Proc. of NIME

[17] http://www.secondlife.com

[18] Liang H, Tay I, Neo MF, Ooi WT, Motani M (2008) Avatar mobility in networked virtual environments: measurements, analysis, and implications. CoRR