Distributed Dynamic Delaunay Triangulation in d-Dimensional Spaces

Gwendal Simon[•] **Moritz Steiner**[⋆◇] Ernst Biersack[◇]

◇ : Institut Eurécom
2229 route des Crêtes
06904 Sophia-Antipolis
France

• : France Telecom - R&D Division
38 rue du Général Leclerc
92794 Issy-Moulineaux Cedex
France

⋆ : University of Mannheim
Computer Science IV
A5, 6 68159 Mannheim
Germany
moritz.steiner@informatik.uni-mannheim.de
Tel: +49 621 181 2614 — Fax: +49 621 181 2601

Abstract

Voronoi diagrams and Delaunay triangulations have proved to be efficient solutions to numerous theoretical problems. They appear as an appealing structure for distributed overlay networks when entities are characterized by a position in a d dimensional space. In this paper, we present some algorithms aiming to maintain an overlay network matching the Delaunay triangulation of the participating entities. We consider that entities are dynamic, that is, they can appear and disappear at any time. We first present the algorithms for a two-dimensional space. Then, we show that these algorithms can be applied in a three-dimensional space. Finally, we generalize them to d-dimensional space.

Keywords: Distributed Computing, Computational Geometry, Peer-to-Peer Systems

This paper is eligible to the Best Student Paper Award.
It contains ten pages. One additional page is devoted to appendix.

1 Introduction

Since their introduction one century ago, Voronoi diagrams [28] and their dual — Delaunay triangulations [5] — have proved to be elegant solutions to many problems [3, 24].

First of all, most of the problems related to Computational Geometry admit clever solutions based on Voronoi diagrams. A first example is the post-office problem. Given the Voronoi diagram of a set of n entities in the plane, the entity closest to an arbitrary point x can be found in time $\mathcal{O}(\log n)$ [17]. Another example is the $k-$nearest neighbors: the k closest pairs of a set of n entities can be computed in time $\mathcal{O}((n+k)\log n)$ [7].

Moreover, the Delaunay triangulation exhibits some suitable properties for networking issues. Thus, a Minimal Spanning Tree is a subgraph of a Delaunay triangulation [26]. Besides, the length of the shortest path in the graph between any pair of entities is at most t times the euclidean distance between these entities. Hence, a Delaunay triangulation is a $t-$spanner with $t \approx 2.5$ [15].

Finally, relevant cluster structures are well reflected by the Voronoi diagrams, which provide efficient solutions to both partitional clustering and hierarchical clustering [19].

Several distributed applications based on shared virtual spaces could benefit from a logical overlay matching the Delaunay triangulation of entities.

For instance, Internet Coordinates systems provide to entities a position in a d-dimensional space according to network characteristics [23, 25]. Some of the most useful functionalities refer to distance problems: determining the closest entity to another one, ranking several entities by distance comparison... A Delaunay-based overlay network may naturally be considered.

Another example are peer-to-peer file sharing systems. Exploiting the semantic proximity between peers appears as an appealing way to improve these systems [27, 18]. Among the challenges, the automatic detection of interest-based communities is often raised. Coupled with an accurate metric, this issue could be achieved by a Voronoi-based overlay network.

Finally, recent works on large-scale Shared Virtual Reality describe fully distributed systems in which entities characterized by a position in a two-dimensional space should be connected to all entities within their virtual surrounding [16, 1, 12]. Among the approaches, a topology based on a Delaunay triangulation [12] exhibits numerous suitable properties. As Voronoi regions tessellate the whole space, any subspace of the virtual world may be seen as the union of Voronoi regions, so may be monitored by the entities whose Voronoi region overlaps the subspace. Therefore, a coherent virtual space can be constructed if entities know the Delaunay triangulation of their local virtual space.

Maintaining a Delaunay-based overlay in a dynamic distributed context has been studied [21, 2]. Unfortunately, the construction of the overlay network relies on an angular feature, which is specific to two-dimensional spaces. Therefore, the algorithms can not be applied to higher dimensional spaces. In [20], an overlay based on a Delaunay triangulation is constructed on a wireless ad-hoc network, but some features of wireless protocols ease the detection of potential neighbors while the algorithms also focus on a two-dimensional space.

In this paper, we describe local algorithms aiming to maintain an overlay network, such that the logical network formed by entities and links between them matches the Delaunay triangulation of the set of entities in the space. We do not restrict the study to two-dimensional spaces. Rather, the algorithms intend to be applied in $d-$dimensional space. Moreover, although addition and deletion of entities in Delaunay triangulation are known to be costly [22, 13], we aim to provide a set of algorithms coping with dynamic entities.

The algorithms should conform to the context of large-scale applications in the Internet. So, entities can appear and disappear at any time, but we assume that entity failures are detected in a reasonable time. Entities only know the entities to which they are connected. That is, no global knowledge is available and all algorithms remain local. Therefore, the overlay network built and maintained by

the algorithms is scalable in the sense that all operations do not depend on the total number of entities participating to the network.

We first present self-organizing algorithms intended for the two-dimensional space. This first part introduces the principle of the algorithms. Then, we describe how these algorithms can be adapted to three-dimensional space. We focus here on exhibiting the correctness of the algorithms applied to higher dimensional space. Finally, we quickly describe the algorithms in $d-$dimensional space.

2 Model and Definitions

An *entity* is a process having communication capabilities. The entities are able to exchange messages through asynchronous reliable bidirectional communication links. The system at time t is modeled by a graph $G(t) = (V(t), E(t))$ where $V(t)$ is the set of entities and $E(t)$ is the set of connections (or edges or links). The set of direct neighbors of an entity e at t is denoted $K(e, t)$.

Each entity is characterized by a *position* in a $d-$dimensional space. The Delaunay triangulation of $V(t)$, noted $DT(t)$, connects entities into non-overlapping $d-$simplices such that the *hypersphere* of each $d-$simplex contains none of the entities in its interior. The hypersphere that passes through all entities of a $d-$simplex T is noted $\mathcal{C}(T)$. We would like to ensure that $\forall t, E(t) = DT(t)$.

We assume in this paper that entities are in *general position*, *i.e.* no $d+1$ entities are on the same hyperplane and no $d+2$ entities are on the same hypersphere. Finally, we consider that the position chosen by a new entity at t is in the interior of the convex hull of $V(t)$.

3 Two-Dimensional Space

In two-dimensional space, the $d-$simplex is usually called a *triangle* while the hypersphere is a *circle*. So, the Delaunay triangulation connects entities such that each triangle contains none of the entities within its circumcircle.

We also use in the following some useful properties of two dimensional spaces. We note $[e, e']$ the line joining the entities e and e' in T. The directed angle $\angle(e'\ e\ e'')$ is defined by the counterclockwise angle formed at e by $[e, e']$ and $[e, e'']$. An entity e_i lies in the directed sector $\nabla(e'\ e\ e'')$ when $\angle(e'\ e\ e'') = \angle(e'\ e\ e_i) + \angle(e_i\ e\ e'')$. Also, we define the successor of an entity e' for an entity e_0 by:

$$s_{e_0} e' = e'' \Leftrightarrow \forall e \in K(e_0) : e \notin \nabla(e'\ e_0\ e'')$$

An entity e is able to easily determine the successor and the predecessor of a neighbor thanks to a list sorted in a counterclockwise order around e. The methods `insert` and `remove` provide the insertion and removal of entities, while `predecessor` and `successor` return respectively the predecessor and the successor of a neighbor.

3.1 Entity Insertion

The insertion of an entity in a Delaunay triangulation has been extensively studied as it is the basis of the *Incremental Construction* algorithm, one of the most robust algorithms for the construction of Voronoi diagrams [10, 11]. Figure 1 illustrates this technique. It consists of (1) finding the triangle enclosing the new entity, then (2) splitting this triangle into three and (3) recursively checking on all adjacent triangles whether the *Edge flipping* procedure should be applied. As shown in Figure 2, the edge flipping algorithm replaces the edge (b, c) by the edge (a, z) if $\mathcal{C}(a, b, c)$ contains z.

We propose to use five distinct messages. Three of them are devoted to the enclosing triangle detection: `find-nearest`, `nearest` and `best`. The two latter are used to open a connection (`hello`) and to notify another entity (`detect`).

We consider a new entity z joining the system at t. The entity z chooses a position and initiates the process by sending a message `find-nearest` to any entity $a \in V(t)$. We first have to detect the closest entity to the position by using a greedy walk algorithm, which is known to always succeed in a Delaunay Triangulation [4].

An entity e replies to a `find-nearest` message by a `nearest` message containing information on

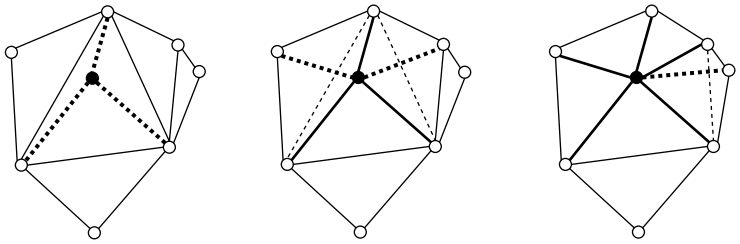

Figure 1: Insertion of a new entity

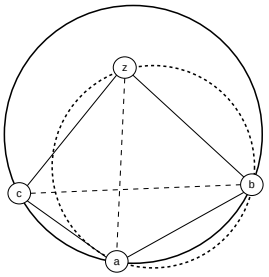

Figure 2: Edge Flipping

the closest entity $e' \in K(e,t)$ to the queried position. If e is closer to the position of z than all of its neighbors, it sends a `best` message. This message contains information on e itself and both the predecessor $e' = s_e^{-1}z$ and the successor $e'' = s_e z$ of z in its neighbor list $K(e,t)$. The entities e, e' and e'' form the triangle enclosing the position chosen by z, which immediately sends a `hello` message to these three entities.

We now consider an entity a receiving a message `hello` from the entity z at time t. In Algorithm 1, the entity a is assumed to have k neighbors. The notation used in the following relies on the situation described in Figure 3.

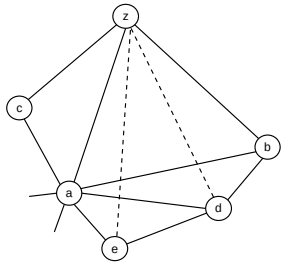

Figure 3: Iterative Edge Flipping

The creation of triangles (a,b,z) and (a,z,c) may have an impact on the edges (a,b) and (a,c). For simplicity, we consider the case of the edge (a,b). The other case is symmetrically identical.

The edge (a,b) is shared by two triangles (a,d,b) and (a,b,z). The entity d is the opposite of z through the edge (a,b) and the predecessor of b in the list of neighbors. The edge flipping mechanism imposes a to remove the edge (a,b) if $\mathcal{C}(a,d,b)$ contains z. In this case, a sends to z a `detect` message containing a description of d. Hence, z is able to contact d by a `hello` message and the edge (z,d) is created. Note that both a and b should test the validity of the edge (a,b). So, both entities implicitly know that the edge (a,b) is discarded. It is possible to imagine an improved version of the algorithm where one edge is only checked by one entity.

Algorithm 1: `Hello z` (received by a)

```
1  insert (z)
2  p_j = successor (z)
3  p_{j-1} = successor (p_j)
4  while z ∈ C(a p_{j-1} p_j) do
5  |   send "detect p_{j-1}" to z
6  |   remove (p_j)
7  |   p_j = p_{j-1}
8  |   p_{j-1} = successor (p_j)
9  p_j = predecessor (z)
10 p_{j+1} = predecessor (p_j)
11 while z ∈ C(a p_j p_{j+1}) do
12 |   send "detect p_{j+1}" to z
13 |   remove (p_j)
14 |   p_j = p_{j+1}
15 |   p_{j+1} = predecessor (p_j)
```

Now, the new entity z may be in conflict with the triangle (a,d,e) where e is the opposite of z through the arc (a,d). If z is within $\mathcal{C}(a,d,e)$, the edge (a,d) should be replaced by an edge (z,e). And so on recursively until the edge flipping algorithm does not switch diagonals containing a.

In Alg. 1, the entity a tests all entities in the clockwise from the successor of z (lines 4-8) until no conflict is detected, then all entities counterclockwise from the predecessor are treated (lines 11-15). The insertion of z is complete when no more message transits in the network.

Lemma 3.1 *A new entity eventually discovers all of its neighbors.*

Proof: We first show that the entity z eventually discovers, in a given sector, all Delaunay neighbors.

We first assume an entity p_i, which, at reception of a message `hello` from a new entity z, detects that the link with its neighbors $p_j \in K(p_i, t)$ should not be flipped with (z, p_{j+1}). It is sure that no new edge will be created in the sector $\nabla(p_i\ z\ p_j)$.

Assume now that all neighbors of p_i should be connected to z. Let p_n and p_{n-1} be the neighbors of p_i such that $p_n = s_z p_i$ and $p_i = s_z p_{n+1}$. It is impossible that there exists an entity p in $\nabla(p_n\ z\ p_{n-1})$ to which z should be connected because this entity p would have to be connected to p_i. So no new edge will be created in $\nabla(p_n\ z\ p_{n-1})$.

The number of neighbors of an entity in a Delaunay triangulation is bounded by $n-1$ where n is the number of entities in system. So, there is at most $n-1$ sectors from which a `detect` message may be issued. When all entities within a given sector has been discovered, the number of sectors not yet completed strictly decreases. Therefore, all sectors are eventually completed. ■

Theorem 3.2 *If the overlay network matches the Delaunay triangulation of $V(t)$, then it will eventually match the Delaunay triangulation of $V(t) \cup \{z\}$.*

Proof: Let $DT(t+1)$ be the Delaunay triangulation of $V(t) \cup \{z\}$. The insertion of z is complete at $t+\gamma$. Two reasons may prevent the set $E(t+\gamma)$ to be similar to $DT(t+1)$: an edge existing in $DT(t+1)$ does not exist in $E(t+\gamma)$ and an edge not existing in $DT(t+1)$ exists in $E(t+\gamma)$. We prove in the following that both cases are impossible.

We assume that an edge (z, p_i) exists in $DT(t+1)$ while it does not in $E(t+\gamma)$. We note p_{i-1} and p_{i+1} the predecessor and the successor of p_i for the entity z. The edge (z, p_i) has not been created if the two entities p_{i-1} and p_{i+1} did not detect that $\mathcal{C}(p_i\ p_{i-1}\ p_{i+1})$ contains z. However, if the edges (z, p_{i-1}) and (z, p_{i+1}) exist, both entities p_{i-1} and p_{i+1} previously received a `hello` message, so they surely detects the edge (z, p_i).

Assume now that there exists an edge e existing in $E(t+\gamma)$ and not in $DT(t+1)$. It is sure that this edge e intersects another edge e'. We consider that $e = (z, p_i)$ and $e' = (p_{i-1}, p_{i+1})$. If the edge (z, p_i) has been created, the entities p_{i-1} and p_{i+1} emitted a `detect` message containing p_i. But, an entity sending a `detect` message also removes the edge in conflict with the new edge. So, it is impossible that (p_{i-1}, p_{i+1}) exists anymore. ■

The *in-circle-test* is the usual name for the test whether an entity belongs to a circumcircle[1]. As this operation is especially costly regarding other computations, it is the main complexity measure in Delaunay triangulation construction.

We show that the total number of *in-circle-tests* required by the insertion of a new entity linked to k neighbors is $4k-6$. But, this task is quite fairly distributed among the neighbors as we also show that the number of *in-circle-tests* performed by one neighbor is less than k.

Another concern is the time complexity. We consider asynchronous communication links, so it is impossible to give a bound on the time required for the insertion of a new entity. However, it is possible to measure the number of *causal operations*: the number of times the new entity should send a `hello` after reception of a `detect` until it discovers all neighbors. We show that the maximal number of causal operations required by the insertion of a new entity linked to k neighbors is less than $\frac{k-3}{2}$.

The proofs of these results are in Appendix A.

3.2 Entity Deletion

We now focus on the deletion of an entity. Entities that leave the system can quickly compute the new connections and inform their neighbors about the new links they have to create. Thus, when the set of neighbors of the faulty entity is known, the optimal computation of the new Delaunay triangulation has a complexity of $\mathcal{O}(k \cdot log\ k)$ where k is the number of neighbors of the entity [6]. Other

[1] $\mathcal{C}(a\ b\ c)$ contains z iff $\begin{vmatrix} a_x & a_y & a_x^2+a_y^2 & 1 \\ b_x & b_y & b_x^2+b_y^2 & 1 \\ c_x & c_y & c_x^2+c_y^2 & 1 \\ z_x & z_y & z_x^2+z_y^2 & 1 \end{vmatrix} > 0$

robust algorithms admit a complexity of $\mathcal{O}(k^2)$ but simpler implementations [22].

But, entities may crash with no graceful behavior. So a neighbor e of a faulty entity z at t does not know its new neighbors in $DT(t+1)$. Fortunately, e can rely on triangulation properties. Especially, it is sure that $K(e,t) \cap K(z,t)$ contains the predecessor and the successor of z in $K(e,t)$.

We consider an entity z crashing at time t in the graph $G(t) = (V(t), E(t))$ with $E(t)$ matching $DT(t)$. We aim to determine $E(t+\delta)$ such that $E(t+\delta) = DT(t+1)$.

We use the usual message `hello` and an additional message `lost`. As soon as an entity e detects the crash of z, it sends a `lost` message to both the predecessor p_i and the successor p_j of the faulty entity in $K(e,t)$. This message contains z and the opposite entity: p_i for the message sent to p_j and *vice versa*. The notation used in the following refers to the situation illustrated by Figure 4.

The `lost` message sent to both neighbors provides the material for the edge flipping. The edge flipping mechanism consists of selecting a diagonal in a quadrilateral. The entity e considers the quadrilateral (f,d,c,e) while the entity d considers (f,d,b,e). There are four obvious cases.

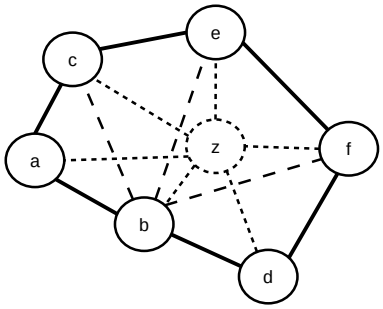

Figure 4: The entity z crashes

At first, neither c, nor b belongs to $\mathcal{C}(f,e,d)$. In this case, the edge (d,e) should be created. Both entities send a `hello` message to each other and the edge is validated. For instance, the edge (b,c) is created with this situation in Fig. 4.

In the opposite case, both c and b are within $\mathcal{C}(f,e,d)$. The entities d and e do not emit any other message than the `lost` message. However, it is sure that only one entity should be connected to f. When the entity f receives the `lost` messages, it tests both situations and it chooses its neighbors in the Delaunay triangulation. For instance, the edge (b,f) is created in this situation in Fig. 4.

Finally, there are two symmetric cases where only one entity belongs to $\mathcal{C}(f,e,d)$. For simplicity, we consider that b belongs to $\mathcal{C}(f,e,d)$ while c does not. In this case, e sends a `hello` message to d, meanwhile d sends a `lost` message to f containing b. An edge is validated only when both entities receive a `hello` message. So, e does not validate the edge (e,d) as it receive no messages from d. On the contrary, when f and b receive the `lost` message from d, they both send to each other a `hello` message such that the edge (b,f) is eventually validated.

Once this process ends, the entities which have a new neighbors — b, c and f in the case of Fig. 4 — re-emit a `lost` message to their neighbors. For instance, f emits a `lost` message to b containing e and one other to e containing b. It leads to the creation of the edge (b,e). This process iterates until no new edge should be created.

Theorem 3.3 *If the overlay network matches the Delaunay triangulation of $V(t)$, then it will eventually match the Delaunay triangulation of $V(t) \setminus \{z\}$.*

Proof: The proof is straightforward. The depicted algorithm resolves the situation for one triangle shared by three neighbors of z. As z was connected at t with a finite number of triangles, the algorithm eventually creates the links between two-hops entities. The number of resulting triangles is strictly less than the initial number of triangles. As it is a recursive process, it eventually builds the Delaunay triangulation of $V(t) \setminus \{z\}$. ∎

4 Three-Dimensional Space

The three-dimensional Delaunay triangulation problem is to connect n entities into non-overlapping *tetrahedra* such that the *circumsphere* of the four vertices of any tetrahedron of the triangulation contains none of the given entities in its

interior [14]. In order to illustrate a Delaunay triangulation in three-dimensional spaces, we propose in Appendix B a stereopsis that may lead to perceive the depth of entities.

In a two-dimensional space, an entity is able to retrieve the Delaunay triangles from the set of its neighbors coupled with the methods `predecessor` and `successor`. Unfortunately the notions of clockwise and counterclockwise have no meaning in three dimensions. Therefore the tetrahedra of the Delaunay triangulation have to be stored explicitly by each entity. We note $\mathcal{T}(e,t)$ the set of tetrahedra associated with the entity e at time t. The circumsphere of a tetrahedron defined by four entities a, b, c and d is noted $\mathcal{C}(a,b,c,d)$.

4.1 Entity Insertion

The insertion of a new entity in three-dimensional space relies on the same principles than in two-dimensional space. First, the nearest entity to the position of the new entity z is determined. Then, the tetrahedron containing z is split. Following, all tetrahedra in conflict with z are detected by the *in-sphere-test*[2]. Finally, a flipping mechanism is used.

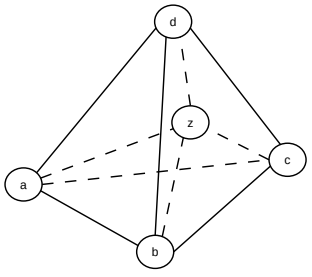

Figure 5: Splitting the enclosing tetrahedron

As in Section 3.1, the detection of the enclosing tetrahedron is managed by the messages `find-nearest`, `nearest` and `best`. This process ends when a message `find-nearest` is received by an entity a which is the closest to the position of z. The entity a has to detect the tetrahedron $T \in \mathcal{T}(a,t)$ enclosing z by the following method. A face $\triangle$ of a tetrahedron T partitions the space into two subspaces. The boolean function $\texttt{same-side}(\triangle, e, z)$ returns `true` if e and z are in the same subspace regarding to $\triangle$[3]. Let $\texttt{opposite}_T(\triangle)$ be the entity in T which does not participate to $\triangle$. An entity z belongs to a tetrahedron T if:

$$\forall \triangle \in T, \texttt{same-side}(\triangle, \texttt{opposite}_T(\triangle), z) = \texttt{true}$$

The entity a successively tests all the tetrahedra in $\mathcal{T}(a,t)$ until it detects T. It then sends to z a message `best` containing T.

As illustrated in Figure 5, a new entity z belonging to a tetrahedron $T = (a,b,c,d)$ splits it into four tetrahedra $T_1 = (a,b,c,z)$, $T_2 = (a,b,d,z)$, $T_3 = (a,c,d,z)$ and $T_4 = (b,c,d,z)$. Note that previous connections between a, b, c and d still remain although the tetrahedron T does not exist anymore.

Following, the new entity z sends a message `hello` to the four entities of T. It contains the tetrahedra that should be created. In the example of Fig. 5, the entity z sends to b a messages `hello` containing the three tetrahedra T_1, T_2 and T_4.

We consider now the entity b receiving the message `hello`. We restrict the study to the tetrahedron $T_1 = (a,b,c,z)$ for simplicity. In the same manner than in Alg. 1, the entity b has to verify whether z is in conflict with any known tetrahedra. In two dimensions, the edge flipping mechanism involves two triangles. In three dimensions, b should consider two tetrahedra as figured in Figure 6. Let e be the entity such that $(a,b,c,e) \in \mathcal{T}(b,t)$. The entity e is the opposite of z through the face (a,b,c). If z belongs to the circumsphere of (a,b,c,e), then both (a,b,c,e) and (a,b,c,z) should be discarded. This is illustrated by the figure on the right. The resulting tetrahedra are $T_{11} = (a,b,z,e)$, $T_{12} = (a,c,z,e)$ and $T_{13} = (b,c,z,e)$. This operation is called the $2-3-$flip.

[2] $p \in \mathcal{C}(a,b,c,d)$ if $\begin{vmatrix} a_x & a_y & a_z & a_x^2+a_y^2+a_z^2 & 1 \\ b_x & b_y & b_z & b_x^2+b_y^2+b_z^2 & 1 \\ c_x & c_y & c_z & c_x^2+c_y^2+c_z^2 & 1 \\ d_x & d_y & d_z & d_x^2+d_y^2+d_z^2 & 1 \\ p_x & p_y & p_z & p_x^2+p_y^2+p_z^2 & 1 \end{vmatrix} > 0$

[3] $\texttt{same-side}((a,b,c),e,z) = \texttt{true}$ if

$$\begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ e_x & e_y & e_z & 1 \end{vmatrix} * \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ z_x & z_y & z_z & 1 \end{vmatrix} > 0$$

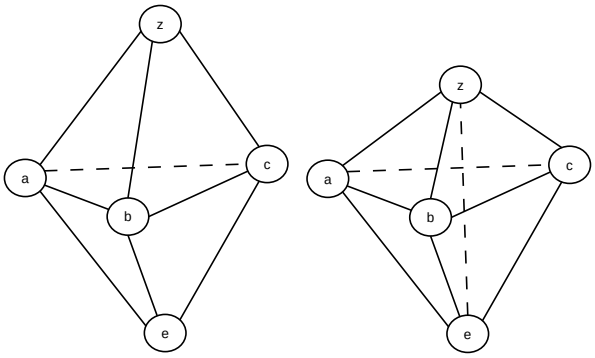

Figure 6: Two tetrahedra may be split into three tetrahedra

In this case, b should immediately inform z that (1) e should be considered as a new neighbors and (2) the tetrahedron (a, b, c, z) should be discarded. It is achieved by a message `detect` containing both e and (a, b, c, z)[4].

Algorithm 2: `hello z` T_1 T_2 T_3 (rec. by a)

```
1  Q.put (T1)
2  Q.put (T2)
3  Q.put (T3)
4  while Q ≠ ∅ do
5      T_a ← Q.pop ()
6      face ← △ ∈ T_a : opposite_{T_a}(△) = z
7      T_b ← T ∈ T(a,t) : face ∈ T
8      if z ∈ C(T_b) then
9          T_i, T_j, T_k ← split (T_a, T_b)
10         Q.put (T_i)
11         Q.put (T_j)
12         Q.put (T_k)
13         e ← opposite_{T_b}(face )
14         send "detect e T_a" to z
15         remove T_b from T(a,t)
16     else
17         insert T_1 in T(a,t)
```

Upon reception of this `detect` message, the entity z removes T_1 from $\mathcal{T}(z, t)$ and adds T_{11}, T_{12} and T_{13} to $\mathcal{T}(z, t)$. Then, it sends a new message `hello` to the entity e with the three new tetrahedra.

Following, the entity b uses the recursive process with both T_{11} and T_{13}. If we consider, for instance, the tetrahedron T_{11}, the entity b first looks for the tetrahedron T_{111} sharing the face (a, b, e) with T_{11}. Then, it verifies whether the new entity z belongs to $\mathcal{C}(T_{111})$. If so, b splits the two tetrahedra T_{11} and T_{111} into three tetrahedra and sends another message `detect` to z.

This process terminates when b does not split any new tetrahedron. The entity b should then decide to close connections with the entities with which it does not share any face of existing tetrahedra.

The pseudo-code of the treatment at reception of a message `hello` is detailed in Algorithm 2. The entity a receives the notification of three new tetrahedra T_1, T_2 and T_3. It puts them on a queue Q managed by a first-in-first-out policy (line 1-3). Then, it retrieves the first tetrahedron, the face that should be tested and the opposite tetrahedron (line 4-6). If the in-sphere-test fails, the new tetrahedron is stored (line 17). Otherwise, the new tetrahedron should be split and the recursive process is achieved by putting the resulting tetrahedra in the queue (line 10-13). In this case, a sends a message `detect` to the new entity (line 14-15).

4.2 Entity Deletion

As in Section 3.2, we consider the abrupt crash of an entity z at time t. None of its former neighbors can know all entities in $K(z, t)$. The algorithm described in the following relies on the same principle as in Section 3.2.

Assume an entity $a \in K(z, t)$ notices the crash of z. The entity a constructs tetrahedra containing itself and all possible combinations — according to the *in-sphere-test* — of entities e belonging to tetrahedra $T_z \in \mathcal{T}(a, t)$ containing z: $e \in T : T \in \mathcal{T}(a, t) \cap \mathcal{T}(z, t)$. For instance, if $\mathcal{T}(a, t)$ contains $T_{z1} = (z, a, b, c)$ and $T_{z2} = (z, a, b, d)$, the entity a builds a new tetrahedron $T_{zn} = (a, b, c, d)$ and removes T_{z1} and T_{z2} from $\mathcal{T}(a, t)$. Finally, the entity a sends a `lost` message to these entities. The message contains the faulty entity z and the new tetrahedra.

We now consider the entity c upon the reception of a `lost` message sent by a. If c has already noticed the crash of z and has already built the tetrahedron

[4]Entities a and c perform the same in-sphere test. So, in a simplified version of the algorithm, it is not mandatory that b notifies a and c that (a, b, c, z) and (a, b, c, e) are not anymore in the Delaunay triangulation.

$T_{zn} = (a, b, c, d)$, the algorithm stops. If T_{zn} is in conflict with an entity $e \in K(c, t)$, a 2-3 flip is done on T_{zn} with e and a `lost` message is emitted for the notification that T_{zn} is not a Delaunay tetrahedron. On the contrary, if c did not detect the failure of z, it should work as a did. That is, it retrieves all tetrahedra $T_z \in \{T \in \mathcal{T}(c, t) : z \in T\}$, then it builds the new tetrahedra from the entities in T_z and sends some `lost` messages to them.

Theorem 4.1 *If the overlay network matches the Delaunay triangulation of $V(t)$, then it will eventually match the Delaunay triangulation of $V(t) \setminus \{z\}$.*

Proof: All entities in $K(z, t)$ eventually notice the crash of z (by a failure detector or by a `lost` message). An entity $a \in K(z, t)$ then removes all tetrahedra containing z from $\mathcal{T}(a, t)$. Thus all tetrahedra containing z are eventually removed from the topology.

Assume now a tetrahedron T_{zn} built by a. If an entity $b \notin K(a, t)$ is in $\mathcal{C}(T_{zn})$, the conflict is not first noticed. But the entity a sends a `lost` message to all entities involving in T_{zn}. If T_{zn} is not a Delaunay tetrahedron, one of the receivers is a neighbor of b. A 2-3 flip is performed on T_{zn} with b and the tetrahedron is not considered anymore. In [9, 29] it is shown, that incremental topological flipping always leads to a correct Delaunay triangulation. ■

5 $d-$Dimensional Space

Three observations may lead to the generalization in $d-$dimensional spaces. First, an entity enclosed in a $d-$simplex splits it into $d + 1$ $d-$simplices. For instance, a $2-$simplex, namely triangle, is split into three triangles and a $3-$simplex — or tetrahedron — is split into four tetrahedra. Secondly, two $d-$simplices can be split into d $d-$simplices. Thus, two triangles generate two triangles while two tetrahedra result in three tetrahedra. This operation is called $2 - d-$flip. Finally, several papers investigates the behavior of the flipping mechanism in $d-$dimensional space. Thus, an incremental algorithm for the triangulation construction is proposed in [29] and, at a later time, the flipping mechanism has been proved to always succeed in constructing the triangulation [9]. Therefore, it is natural to generalize the algorithm previously described.

We begin by generalizing the first part of the algorithm. The closest entity to the position chosen by the new entity z is eventually reached after successive exchanges of `find-nearest` and `nearest`. This entity determines the $d-$simplex T enclosing z and sends to z a `best` message containing the description of T. After the reception of the `best` message, the new entity z splits the simplex T into $d+1$ non-overlapping $d-$simplices and sends a message `hello` to its $d + 1$ new neighbors.

We then generalize the recursive process. We consider the entity b at reception of a message `hello` containing a $d-$simplex T_1. Let $\diamond$ be the $(d - 1)-$simplex generated by all the entities involved in T except the new one z. The entity b can retrieve the $d-$simplex T_2 which shares $\diamond$ with T_1. The *in-hypersphere-test* is applied on T_2. If the entity z belongs to $\mathcal{C}(T_2)$, the entity b sends a message `detect` to z. The edge flipping mechanism results in discarding T_1 and T_2 and creating d $d-$simplices $T_{11}, T_{12} \ldots T_{1d}$.

Finally, as described in Sec. 4.1, the entity b iteratively tests all the $d-$simplices until no new $d-$simplex is created.

In the same manner, the Entity Deletion algorithm in d dimensions is very similar to the $3-$dimensional one.

The entity z crashes at t and none of its former neighbors has connections to all of entities of $K(z, t)$. We consider an entity $a \in K(z, t)$ which notices the crash of z. It first retrieves the $d-$simplices $T_z \in \{T \in \mathcal{T}(a, t) : z \in T\}$. Then the entity a builds — according to the *in-hypersphere-test* — all $d-$simplices from the set of entities involving in T_z. Assume, for instance, that z and a were linked through the d-simplices $T_{z1} = (z, a, b, e_0, \ldots, e_{d-4}, c)$ and $T_{z2} = (z, a, b, e_0, \ldots, e_{d-4}, d)$. The crash of z leads a to remove T_{z1} and T_{z2} from $\mathcal{T}(a, t)$ and to build a new d-simplex $T_{zn} = (a, b, e_0, \ldots, e_{d-4}, c, d)$.

Following, the entity a sends a message `lost` to

the entities involving in the new $d-$simplices. This message contains the faulty entity z and the new $d-$simplices. Upon the reception of a `lost` message, the entity c performs the *in-hypersphere-test* on T_{zn} with every entity e lying on the opposite of the $(d-1)$-simplex of T_{zn}. The new $d-$simplex is kept if the test fails while a $2-d$ flip is applied on T_{zn} if e belongs to $\mathcal{C}(T_{zn})$.

6 Conclusion

In this paper, we present a set of algorithms that dynamically maintain a distributed overlay network that matches the Delaunay triangulation of the entities. The entities can have a position in any $d-$dimensional space. We consider here the arrival of new entities and the crash of an entity. We show that the algorithms we propose succeed in reconstructing the Delaunay triangulation through a self-stabilization scheme. The algorithms proposed for three and d dimensions are very similar, whereas in 2-dimensions it is possible to rely on an angular feature, that is not available in higher dimensions.

We intend to use the protocol in $3-$dimensions for shared virtual worlds. Especially, we may apply it in the Solipsis [16] platform.

One of the main drawbacks of these algorithms is the greedy walk needed to detect the entity closest to the queried position. Some recent studies [8] show how to construct small-world networks by adding only one edge between two entities in the overlay. In these small-world networks, a basic walk is guaranteed to succeed in polylogarithmic time. We will try to transform the Delaunay triangulation to a small-world network, such that the detection of the closest entity could be substantially more efficient.

Acknowledgment

We would like to thank Antoine Pitrou who first had the intuition of the deletion algorithm in two-dimensional space and Joaquín Keller who encouraged and supported this study.

References

[1] A.Bharambe, S. Rao, and S. Seshan. Mercury: a Scalable Publish-Subscribe System for Internet Games. In *Workshop on Network and System Support for Games (Netgames'02)*, 2002.

[2] F. Araujo and L. Rodriguez. Geopeer: A location-aware peer-to-peer system. Technical report, Faculdade de Ciências da Universidade de Lisboa, Portugal, 2001.

[3] F. Aurenhammer and R. Klein. Voronoi Diagrams. In *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers, 2000.

[4] P. Bose and P. Morin. Online Routing in Triangulations. In *International Symposium on Agorithms and Computation*, 1999.

[5] B. Delaunay. Sur la sphère vide. *Otdelenie Matematicheskii i Estetvennyka Nauk*, 7:793–800, 1934.

[6] O. Devillers. On Deletion in Delaunay Triangulations. In *Symp. on Computational Geometry*, 1999.

[7] M. T. Dickerson, R. L. Drysdale, and J-R. Sack. Simple Algorithms for Enumerating Interpoint Distances and Finding $k-$Nearest Neighbors. *Int. Journal of Computational Geometry and Applications*, 2:221–239, 1992.

[8] P. Duchon, N Hanusse, E. Lebhar, and N. Schabanel. Could Any Graph Be Turned Into a Small World. Technical Report 62, Laboratoire de l'Informatique du Parallèlisme (LIP), 2004.

[9] H. Edelsbrunner and N. R. Shah. Incremental Topological Flipping Works for Regular Triangulations. *Algorithmica*, 15:223–241, 1996.

[10] P. Green and R. Sibson. Computing Dirichlet Tesselations in the Plane. *The Computer Journal*, 21:168–173, 1978.

[11] L. Guibas, D. Knuth, and M. Sharir. Randomized Incremental Constructions of Delaunay and Voronoi Diagrams. *Algorithmica*, 7:381–413, 1992.

[12] S-Y. Hu and G-M. Liao. Scalable Peer-to-Peer Networked Virtual Environment. In *Network and Systems Support for Games (NetGames'04)*, 2004.

[13] C. Icking, R. Klein, P. Koellner, and L. Ma. Java Applets for the Dynamic Visualization of Voronoi Diagrams. *Computer Science in Perspective, Springer-Verlag*, 2003.

[14] B. Joe. Construction of Three-Dimensional Delaunay Triangulations Using Local Transformations. *Computer Aided Geometric Design*, 8:123–142, 1991.

[15] J. M. Keil and C. A. Gutwin. Classes of Graphs Which Approximate the Complete Euclidean Graphs. *Discrete and Computational Geometry*, 7:13–28, 1992.

[16] J. Keller and G. Simon. Solipsis: A Massively Multi-Participant Virtual World. In *International Conference on Parallel and Distributed Techniques and Applications (PDPTA'03)*, 2003.

[17] D. G. Kirkpatrick. Optimal Search in Planar Subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[18] F. Le Fessant, S. Handurukande, A-M. Kermarrec, and L. Massouli. Clustering in Peer-to-Peer Sharing Workloads. In *International Workshop on Peer-to-Peer Systems (IPTPS'04)*, 2004.

[19] I. Lee and V. Estivill-Castro. Polygonization of Point Clusters through Cluster Boundary Extraction for Geographical Data Mining. In *Proceedings of the 10th International Symposium on Geospatial Theory, Processing and Applications*, 2002.

[20] X.Y. Li, G. Calinescu, and P-J. Wan. Distributed Construction of a Planar Spanner and Routing for Ad Hoc Wireless Networks. In *Proceedings of IEEE Infocom*, 2002.

[21] J. Liebeherr and M. Nahas. Application-layer multicasting with delaunay triangulation overlays. *IEEE Journal on Selected Areas in Communications*, 20(8):1472–1488, October 2002.

[22] M-A. Mostafavia, C. Gold, and M. Dakowiczb. Delete and Insert Operations in Voronoi/Delaunay: Methods and Applications. *Computers & Geosciences*, 29:523–530, 2003.

[23] T. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM'02*, 2002.

[24] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tesselations: Concepts and Applications of Voronoi Diagrams*. John Wiley and Sons, 2000.

[25] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Virtual Landmarks for the Internet. In *International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.

[26] M. I. Shamos and D Hoey. Closest-Point Problems. In *IEEE Symposium on Found. Comput. Sci. (FOCS'75)*, pages 151–162, 1975.

[27] K. Sripanidkulchai, B.Maggs, and H. Zhang. Efficient Content Location using Interest-Based Locality in Peer-to-Peer Systems. In *INFOCOM'03*, 2003.

[28] G. Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die Reine and Angewandte Mathematik*, 133:97–178, 1908.

[29] D. F. Watson. Computing the n-Dimensional Delaunay Triangulation with Application to Voronoi Polotypes. *The Computer Journal*, 24(2):167–172, 1981.

A Complexity Analysis

Lemma A.1 *The total number of in-circle-tests required by the insertion of a new entity linked to k neighbors is* $4k - 6$.

Proof: Each neighbor of the new entity, except the three entities in the enclosing triangle, has been detected by its two neighbors. Each detection requires one *in-circle-test*, so $2 * (k - 3)$ operations should be performed. Moreover, the insertion requires 2 additional *in-circle-test* by entity before to end the algorithm. So, $k*2$ additional *in-circle-test* are necessary. Therefore, $2*k+2*(k-3) = 4*k-6$ operations are realized. ■

Lemma A.2 *The number of in-circle-tests performed by one neighbor of a new entity linked to k neighbors is less than k.*

Proof: Let z be the new entity and $\{p_0, p_1, \dots p_i, \dots p_k\}$ the set of neighbors of z in $DT(t + 1)$. The worst case is as follows: (p_0, p_1, p_k) is the triangle enclosing z and all entities in $\nabla(p_1\ z\ p_k)$ are discovered by one unique entity (either p_1 or p_k). In this case, this entity should realize $k - 3$ times the *in-circle-test* and two additional *in-circle-test* for the end of the algorithm. ■

Lemma A.3 *The number of causal operations required by the insertion of a new entity linked to k neighbors is less than* $\frac{k-3}{2}$.

Proof: Let z be the new entity and $\{p_0, p_1, \dots p_i, \dots p_k\}$ the set of neighbors of z in $DT(t + 1)$. The worst case occurs when the length of the longest path between z and the farthest entity is maximal. That is, this situation occurs when the enclosing triangle is (p_0, p_1, p_k). In this case, the farthest entity is the entity which lies on the middle of the path between p_1 and p_k, so the entity $p_{\frac{k-1}{2}}$. The three first entities are discovered without any causal operations. On the contrary, all following detected neighbor requires one causal operation, so at worst $\frac{k-3}{2}$ operations if k is odd. ■

B Three Dimensional Stereopsis

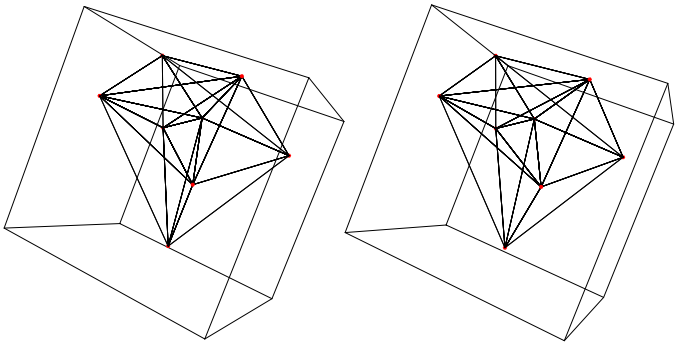

Figure 7: Naked eye inversion stereopsis diagram of a Delaunay triangulation in a three-dimensional space.