University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Construction of Geometric Models for Moving Points

The State of the Art and Concept of Ph.D. Thesis

Tomáš Vomáčka

Technical Report No. DCSE/TR-2010-01
May, 2010

Distribution: public

Technical Report No. DCSE/TR-2010-01
May 2010

Construction of Geometric Models for Moving Points

Tomáš Vomáčka

Abstract

Geometric models for moving points represent a valuable tool of modern computational geometry. In this work, we focus mainly on kinetic Delaunay triangulation which is a special case of the ordinary (static) Delaunay triangulation intended to be used together with moving points without losing its abilities. The first part of the following text describes the overall properties of Delaunay triangulation and the algorithms which are most commonly used for its construction. The following part describes the process of kinetization of the static data structure (together with the properties of the kinetic model). In the next part, we show some of the applications of kinetic Delaunay triangulation, such as collision detection, simulation of crowds, mathematical simulations in the field of fluid dynamics and motion interpolation. Finally, we present our own method for kinetic Delaunay triangulation management together with some new theoretical findings and two applications – kinetic Delaunay triangulation used as a tool for video representation and as a core part of an early warning system for air traffic control.

This work was supported by Project No. GAČR 201/09/0097.

Contents

Chapter 1

Introduction

In order to be able to handle a set of moving data (in our case, these are most commonly represented by moving points), one has to upgrade the commonly used spatial division data structures so that they become able to incorporate the movement. There are several ways of doing so – the most straighforward approach is to discretize the movement and exploit such a set of tools that allow both adding and removing the points into and from the data structure. The movement is then simulated by removing the points from the structure and reinserting them back on new positions according to their trajectories. Such structures are then called *dynamic*. The other commonly used approach is based on the geometric features specific to the data structure. By computing the times when the data structure reaches a singular state due to the movement of the points, we are able to maintain its properties by introducing some kind of local geometry updates. The structures of this kind are called *kinetic*. In my work, I will almost exclusively focus on the kinetic data structures.

As said before, the kinetic data structures represent a special kind of the usual spatial division data structures modified so that they can handle movement of the given generator set. Similarly to the ordinary planar spatial division methods, the given space (usually the Euclidean plane) is divided into cells according to a certain set of generators and a certain set of rules. The generators are in our case given as a set of points, but sometimes it is convenient to use more sophisticated generators such as weighted points, line segments, circles, general polygons or even more complex primitives. The construction rules determine the type and properties of the spatial division and most commonly take the form of minimizing a given function. The most commonly used kinetic data structures are represented by kinetic Voronoi diagram and especially by its dual structure – kinetic Delaunay triangulation – and their modifications (detail on these data structures may be found in [31, 40]).

The cells produced by using Voronoi diagram are composed of points that are closer to their generator (each cell belongs to one of the generators) than to any other primitive in the generator set. Together with the motion of the points inside the generator set, this basic feature is most commonly used in applications where general location or proximity of the generators plays an important role. Such applications may include collision detection (as in [22]), navigation in the virtual environments (shown in [8, 23]), mesh generation for various purposes, such as fininte element method (see [7]) and many others. As said before, the Delaunay triangulation is a structure dual to Voronoi diagram and thus have exactly the same features. Moreover, the triangles produced by the Delaunay triangulation are usually of very good quality (close to equillateral, prolonged and narrow triangles occur rarely) which makes it very useful for various applications where a triangular mesh needs to be generated and its quality is important.

My research is focused mostly on the problematics of kinetic Delaunay triangulations – their features and various applications, however I will also briefly adress the problematics of other commonly used types of kinetic data structures. This work is organized as follows: Chapter 2 describes the spatial division structures that are most commonly used together with kinetic data, Chapter 3 provides an overview of the construction algorithms most suitable for the construction of the aforementioned data structures (with respect to the movement in the dataset). Chapter 4 describes the algorithms used for managing the movement of the points, Chapter 5 shows some of the applications of the kinetic data structures, Chapter 6 describes our contributions to the problematics, Chapter 7 outlines my future work and Chapter 8 concludes this report.

Additional information about my research on this topic, together with some demonstration video sequences, may be found at the homepages of this project which is located at the following adress: `http://graphics.zcu.cz/Projects/Kinetic-triangulation`

Chapter 2

Spatial Division Types

In this chapter, we will describe the most common spatial division data structures that are being used together with moving data. These data structures will be only seen from the static point of view; the exact way of handling the movement in the generator set will be shown in Chapter 4.

Note. In computational geometry, we deal with many different data structures. These structures are usually defined by functions of the input data (e.g., the coordinates of the given points). We call these functions *predicates*) and they very often take the form of defining the sign of a matrix determinant. Some of the common predicates used in computational geometry will be shown later in this chapter and will be referred to in Chapter 4.

2.1 Voronoi Diagram

Let $S = \{p_1, \dots, p_n\}$ be a set of n points in E^2, then let us define:

Definition 2.1.1 (Voronoi cell)**.** A set of points $V(p_i)$, where:

$$V(p_i) = \{x \in E^2 \mid \forall p_j \in S : i \neq j : \|p_i - x\| \leq \|p_j - x\|\} \tag{2.1}$$

is called a Voronoi cell of p_i (point p_i is called the generator of $V(p_i)$).

Definition 2.1.2 (Voronoi diagram)**.** A set of Voronoi cells

$$VD(S) = \{V(p_1), \dots, V(p_n)\} \tag{2.2}$$

is called a Voronoi diagram of S (S is called the set of generators of $VD(S)$).

The boundaries of the Voronoi cells are also called the *Voronoi edges* and are composed of such points that are equally distant to two of the generators. Points of conjuction of the Voronoi edges are called *Voronoi vertices* and represent points that are equally close to three or more of the generators. An example of a Voronoi diagram is shown in Fig. 2.1

2.2 Delaunay Triangulation

A planar triangulation $T(S)$ of set of n points $S = \{p_1, \dots, p_n\}$ may be defined in the following fashion:

Definition 2.2.1 (Planar triangulation [31, 56])**.** A set of edges in E^2 that fulfill the following conditions:

- No two edges $E_1, E_2 \in T(S)$ intersect at a point not in S.
- The edges divide the convex hull of S into triangles.
- The spatial division of the convex hull is maximal.

Now we may define the Delaunay triangulation as a special case of a planar triangulation:

Definition 2.2.2 (Delaunay triangulation [31])**.** Triangulation $T(S)$ that fulfills the condition:

- No point $p_i \in S$ lies inside a circumcircle of any of the triangles in $T(S)$.

is called *Delaunay triangulation* of S, or $DT(S)$.

The added condition is also sometimes called the Delaunay condition or the empty circumcircle condition. An example of a Delaunay triangulation, together with a Voronoi diagram is shown in Fig. 2.1. In this figure, the duality between these two data structures may be clearly seen – for each point in the generator set, there is a cell in $VD(S)$. The number of edges this cell has is equal to the number of other points in the generator set, to which is this generator connected (i.e., for each edge separating two generator points in the Voronoi diagram, there is a corresponding edge connecting these two points in the Delaunay triangulation).

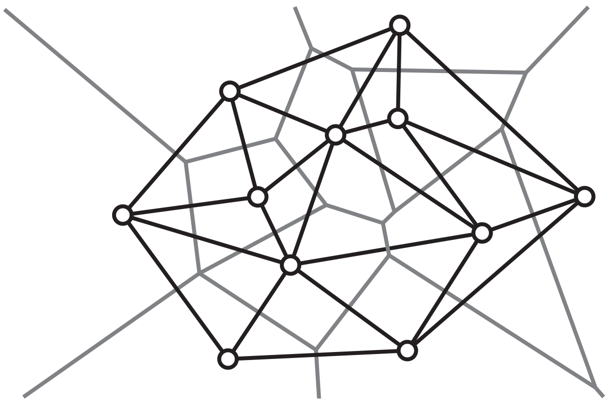


Figure 2.1: Voronoi diagram (grey lines) and its dual structure, the Delaunay triangulation (black lines).

This duality between Delaunay triangulation and Voronoi diagram may be used for various purposes. The most straighforward use is represented by the possibility to construct one of these structures from the other (e.g., construct $VD(S)$ from a given $DT(S)$ or vice versa). However, this case is used rarely, because it is often more convenient to create the wanted structure directly. On the other hand, if we are given a set of points and want to exploit some of the properties of a Voronoi diagram over this set, we will often construct $DT(S)$ rather than the Voronoi diagram. This is caused by the fact, that the Delaunay triangulation is composed of only one type of primitives (triangles versus general convex polygons) and thus it is often much simpler to construct and maintain.

2.2.1 Properties of Delaunay Triangulation

Uniqueness and Singular Configurations

As long as there are no four cocircular points in the generator set, the Delaunay triangulation will be unique for the given set. This property is caused by the feature that defines the Delaunay triangulation (no point may lie inside a circumcircle of any triangle in DT) – if there are four (or more) cocircular points in the generator set, then there will be two possible legal triangulations that fulfill all the given criteria, see Fig. 2.2 – these points are then said to be in a *singular state*. It is important to note that the singular states are crucial for handling the movement of the generating points (see further).

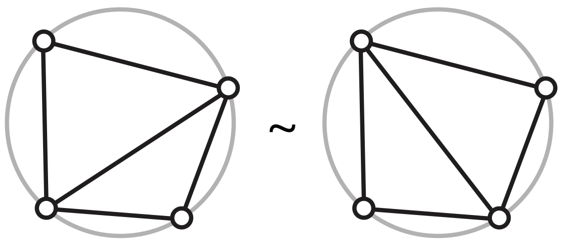


Figure 2.2: Two legal topologic configurations in Delaunay triangulation.

Locality

As shown by Delaunay himself in [11], any triangulation which is locally Delaunay, is also globally Delaunay. This means that if each pair of adjacent triangles in the triangulation fulfills the abovementioned Delaunay condition (i.e., if we construct the circumcircle of arbitrary one of the two adjacent triangles, the remaining point of the quadruple will not lie inside this circle), then each point in the triangulation fullfills this condition against each of its triangles.

Higher Dimension Embedding

Let us first define some of the terms we will need in order to inspect the relationship between Delaunay triangulations in E^2 and convex hulls in E^3:

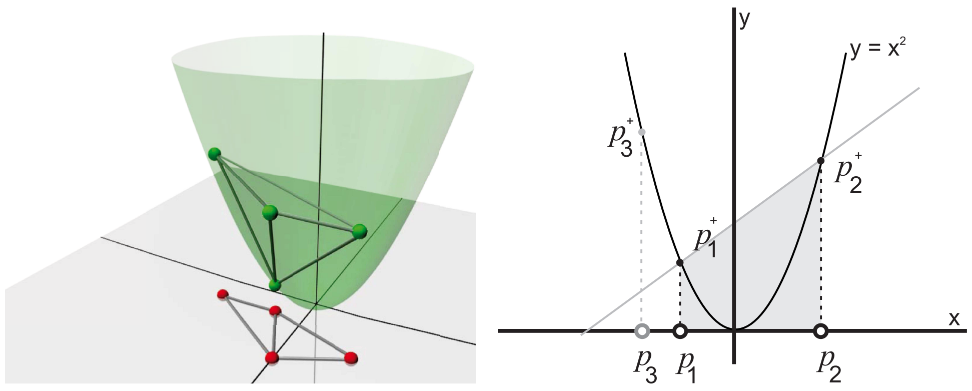
Definition 2.2.3 (Lifted point, lifted space)**.** Given a point $p = (x, y)$ in E^2, its lifted point $p^+ \in E^3$ is given as:

$$p^+ = (x, y, x^2 + y^2) \tag{2.3}$$

and similarly for higher dimensions. If p lies in a $d-$dimensional Euclidean space, p^+ will lie on a paraboloid in a $(d + 1)-$dimensional space, called *lifted space*.

It is well known (see [16, 31]) that Delaunay triangulations (as well as Voronoi diagrams) are closely related to higher dimension convex hulls. Given a set of points S and a set of corresponding lifted points $S^+ = \{p_1^+, p_2^+, \ldots, p_n^+\}$, the Delaunay triangulation $DT(S)$ will be equal to planar projection of lower $CH(S^+)$.

In other words, if we project the points in the generator set on a paraboloid $z = x^2 + y^2$, construct a convex hull of this projection and project the lower facets of this convex hull back onto the original plane using a planar projection, we obtain a Delaunay triangulation of the original point set. An illustration of this relationship is given in Fig. 2.3(a).



(a) 2D triangulation example (b) 1D illustrational example

Figure 2.3: Relationship between a convex hull and a Delaunay triangulation.

We can see in this figure a set of points in E^2 represented by the red balls. These points are then projected on a paraboloid using the relation in Eq. 2.3 (the projections are markded by the green balls). A convex hull constructed over the set of the projections is shown and its lower facets are then projected back into E^2 where they form the Delaunay triangulation of the original (red) points.

Nearest Pair

As said before, the Voronoi edges of $VD(S)$ are composed entirely of points that are equally distant to two of the generators. Considering the duality between $VD(S)$ and $DT(S)$ we may see that in the $DT(S)$, each of the generators is connected to several other nearby points. The exact number of such connections depends on the exact locations of each of the generators, but each generator will be connected to its nearest neighbor.

2.2.2 Incircle Test

As stated before, the Delaunay criterion ensures that no point $p_i \in S$ in the generator set lies inside a circumcircle of any of the triangles in $DT(S)$. There are numerous ways to determine whether a point lies inside a circumcircle of a given triangle, but the most commonly used one is based on the relationship between the Delaunay triangulation and the higher dimension convex hulls which we described earlier. An illustration is shown in Fig. 2.3(b), where we may see a one-dimensional illustration of the situation. The lifted points p_i^+ are projected on a parabola. The 1D variant of the incircle test is then in fact represented by an orientation test – in order to determine if p_3 lies between p_1 and p_2 we have to check if p_3^+ lies below the line defined by p_1^+ and p_2^+. The situation in higher dimensions is then very simillar.

Mathematically, we can formulate the Delaunay criterion in the form of a matrix determinant computation. Given a planar triangle $p_i p_j p_k$ and a point p_l, we may determine if p_l lies inside the circumcircle of $p_i p_j p_k$ by computing the value of *incircle*:

$$incircle = \det \mathbf{I} = \det \begin{bmatrix} x_i & y_i & x_i^2 + y_i^2 & 1 \\ x_j & y_j & x_j^2 + y_j^2 & 1 \\ x_k & y_k & x_k^2 + y_k^2 & 1 \\ x_l & y_l & x_l^2 + y_l^2 & 1 \end{bmatrix} \tag{2.4}$$

where $p_i = (x_i, y_i)$ and so forth for p_j, p_k and p_l.

If the triangle defined by points $p_i p_j p_k$ is oriented counterclockwise, a positive value of (2.4) means that p_l lies inside the circumcircle of $p_i p_j p_k$, a negative value means that p_l lies outside and a zero value shows that the point lies exactly on the circumcircle. If the orientation of the points $p_i p_j p_k$ is unknown, an orientation test needs to be made together with the incircle test (see [31] for details):

$$orient = \det \mathbf{O} = \det \begin{bmatrix} x_i & y_i & 1 \\ x_j & y_j & 1 \\ x_k & y_k & 1 \end{bmatrix} \tag{2.5}$$

We may then compute the *incircle* value as follows:

$$incircle = incircle' \cdot orient \tag{2.6}$$

where *incircle′* is in the exact same form as shown in (2.4) with the exception that it is not necessary to know the orientation of $p_i p_j p_k$ beforehand.

Note that the incircle test function is an example of a predicate which has been mentioned at the beginning of this chapter as it is a function of the input data and it is the only function we need to compute in order to determine if a triangulation is Delaunay or not.

2.3 Regular Triangulation and Power Diagram

Regular triangulation and its dual structure, the power diagram (see [17]), represent a generalization of the Delaunay triangulation and Voronoi diagram. The principle of this generalization lies in the fact, that instead of the ordinary points, we use a set of weighted points as generators (e.g., each of the points $p_i \in S$ is assigned a real number w_i called *weight*). The weight of the points is then used to compute the distance of the generating points (it is then called power distance):

Definition 2.3.1 (Power distance)**.** Given a weighted point $p \in E^2$ with weight w_p and a point $x \in E^2$ (which may be weighted, but it is not necessary), we compute the power distance $\pi_p(x)$ between p and x as follows:

$$\pi_p(x) = ||px||^2 - w_p \tag{2.7}$$

where $||px||$ is the Euclidean distance between p and x.

The power distance may be seen as a square distance of a tangent between x and a circle with center in p and radius of $\sqrt{w_p}$ – see Fig. 2.4.

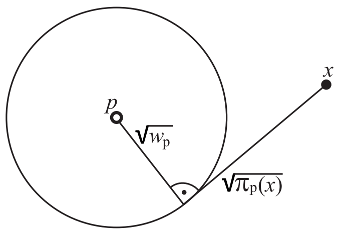


Figure 2.4: Power distance of two points.

Definition 2.3.2 (Orthogonal points)**.** Two points p_i and p_j are said to be orthogonal if $\|p_i p_j\| = w_i + w_j$

An example of two orthogonal points is shown in Fig. 2.5

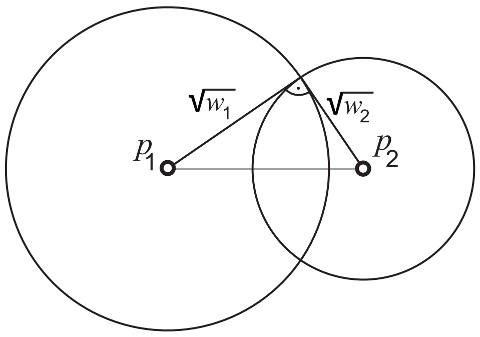


Figure 2.5: Two orthogonal points.

Redundant Points

According to [17, 31], some points may occur in the generator set that will not become vertices in the final triangulation. These points are then called *redundant points*. This feature has a close connection to the relationship between the triangulations and higher dimension convex hulls that was shown before.

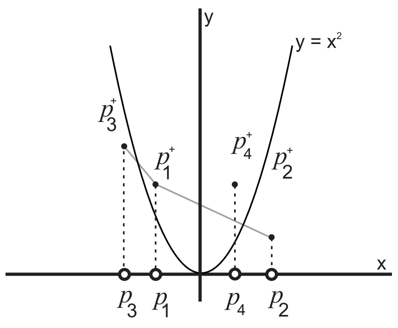


Figure 2.6: Weighted point projection on the paraboloid.

As we can see in Fig. 2.6, the lifted versions of the weighted points are not projected directly onto the paraboloid (which is the case in Delaunay triangulation) but their vertical coordinate is altered by their weight, or more precisely – the weight of each of the points is substracted from its projected coordinate, thus gaining lifted points with coordinates $p^+ = (x_i, y_i, x_i^2 + y_i^2 - w_i)$ rather than the ones shown in Eq. 2.3. The result of this alternation is then that there may occur some points that do not lie on the lower convex hull (as is the case of p_4^+ in this figure which lies above the lower convex hull).

Power Incircle Test

Due to the difference in the computation of the distance between two points, there is also a change in the computation of the incicrcle test for the purposes of the regular triangulations: (2.4) has to be modified in order to include the weights of the generating points.

$$incircle_{RT} = \det \mathbf{I} = \det \begin{bmatrix} x_i & y_i & x_i^2 + y_i^2 - w_i & 1 \\ x_j & y_j & x_j^2 + y_j^2 - w_j & 1 \\ x_k & y_k & x_k^2 + y_k^2 - w_k & 1 \\ x_l & y_l & x_l^2 + y_l^2 - w_l & 1 \end{bmatrix} \tag{2.8}$$

Singular Point Configurations

As long as the weights assigned to the points in S are of the same value (it does not matter if this value is zero or nonzero), the topology of the regular triangulation will be exactly the same as if we constructed an ordinary Delaunay triangulation for the given set of (non-weighted) points. If the weights of the points are different, singular states will occur for quadruples of orthogonal points. A singular case in regular triangulation is shown in Fig. 2.7, where both available triangle pairs form a regular triangulation.

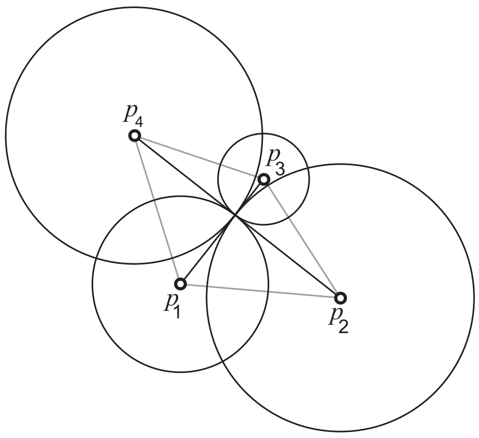


Figure 2.7: Singular case in a regular triangulation.

Chapter 3

Construction Algorithms

There are many different algorithms that are commonly used in order to create various types of spatial division data structures. This chapter provides a brief overview of the most commonly used construction methods together with the data structures mentioned in the previous chapter. Furthermore, a more detailed description of the incremental insertion algorithm is given, because its properties make it extremely suitable for the given purpose.

3.1 Construction Method Properties

There are numerous properties that may be used to compare the construction algorithms. Among them, the most commonly used are the following (for others, see [9]):

Time and space complexity: the algorithms are often judged by the amount of time and space they need in order to function. Perhaps the most important feature of each algorithm is the dependency of the runtime it consumes on the size of the input dataset, but the amount of consumed memory is not less important, especially for algorithms that are designed to operate on large datasets.

Online property: we say that an algorithm is online if it allows addition of points into the set of generators after the initial construction step.

Parallelism: the possibility to parallelize an algorithm represents a valuable possibility to increase its performance, however some of the mentioned algorithms are unsuitable for parallel execution.

Extensibility to higher dimensions: even though our research only uses planar data and thus higher dimension functionality is of rather low importance for us, some of the described algorithms may not be extended to higher dimensions which may handicap them if this kind of a future development is considered.

3.2 Construction Methods Overview

Higher Dimension Embedding: as shown before, there is a relation between $n-$dimensional Delaunay triangulations and $n+1-$dimensional convex hulls which may be used for the purposes of DT construction (see [4] for details). It is obvious that the properties of this algorithm depend solely on the used method for convex hull construction. The time complexity is in the optimal case equal to $O(n \log n)$.

Divide and Conquer: this type of algorithms is based on the idea that the initial dataset may be divided recursively – partial *sub*-triangulations are then created for the divided parts and connected together (see [9]). Divide and conquer algorithms are usually time optimal in the worst case, even although they are usually untrivial to implement. This approach is not online, easy to parallel but the extensibility to higher dimensions is difficult because of problems with sorting in higher dimensions. The time complexity of this approach is $O(n \log n)$ in the worst case.

Local Optimization: starting with a random initial triangulation of the given dataset, we may exploit the local optimality of Delaunay triangulation. By testing the satisfaction of the Delaunay property for each pair of convex adjacent triangles, we discover the locations where the Delaunay property is violated and convert the given triangulation to DT by flipping the common edge of these triangle pairs. This method is not online, may not be reliably extended to three dimensions, is difficult to parallelize and its performance is dependent on the algorithm used for creating the initial construction. The time complexity of this approach is determined by the number of edge flips we need to perform. In E^2, it is $O(n^2)$ in the worst case and $O(n)$ expected. Details may be found in [31, 34].

Incremental Construction: Delaunay triangulation may be constructed by starting with the shortest edge that may be constructed from the given dataset and successive construction of such triangles, that fulfill the Delaunay condition (see [12]). The time complexity of this approach is $O(N^3)$ in two dimensions if we do not use any acceleration data structures.

Sweep Construction: after sorting the given points along an axis, we create the Delaunay triangulation by adding them to a partial triangulation in the order given by the sorting. This approach is not online, may be parallelized, is extensible to three dimensions and runs in $O(n \log n)$ time. Examples of this approach may be found in [21, 47]

Incremental Insertion: starting with an initial triangle large enough to contain the given dataset, we add the points from the dataset one at a time, divide the triangle that contains the added point and locally repair the triangulation if the Delaunay condition is broken in the process. This algorithm will be described in detail in the following section.

3.3 Incremental Insertion Algorithm Details

3.3.1 Overall Functionality

The functionality of the incremental insertion algorithm (presented in [32]) for constructing a Delaunay triangulation is shown in Alg. 3.1. As we can see, the algorithm consists of four distinct parts – first, the initial simplex is constructed that contains the whole area covered by the points in S, after that, the points from the generator set are separately added to the current triangulation (which contains only the auxiliary simplex at the beginning of this step) and during the addition of the points, the triangulation is being constantly checked for edges that do not satisfy the Delaunay condition and these edges are replaced by the other possible edges by using flipping. This process is called edge legalization. Finally, all the edges connecting the points from the auxiliary simplex with any other point are removed from the triangulation (this step is usually not performed in the kinetic applications, see further).

```
Input: Set $S = \{p_1, p_2, \ldots, p_n\}$ of $n$ distinct points
Output: $DT(S)$
$DT(S) = p_0 p_{-1} p_{-2}$ containing $S$                              // Initialization

foreach $p_r \in S$ do
    Localize triangle $p_i p_j p_k$ such that $p_r \in p_i p_j p_k$
    if $p_r$ is inside $p_i p_j p_k$ then
        $DT(S) = DT(S) \setminus \{p_i p_j p_k\}$
        $DT(S) = DT(S) + \{p_r p_i p_j, p_r p_j p_k, p_r p_k p_i\}$
        // $p_i p_j p_k$ is split into three triangles
    else                  // $p_r$ lies on the edge common to $p_i p_j p_k$ and $p_i p_j p_l$
        // The two adjacent triangles are split into four
        $DT(S) = DT(S) \setminus \{p_i p_j p_k, p_i p_j p_l\}$
        $DT(S) = DT(S) + \{p_r p_i p_k, p_r p_j p_k, p_r p_l p_i, p_r p_l p_j\}$
    end
    Legalize all newly created edges
end
Remove all the edges containing $p_q \in \{p_0, p_{-1}, p_{-2}\}$ if needed
```

Algorithm 3.1: Overall functionality of the incremental insertion algorithm for DT construction

3.3.2 Initial Triangle Construction

In order to be able to locate the point location performed in the following step, we need to make sure that a triangle containing the added point exists. This can be done by encapsulating the whole area covered by the triangulation by a single simplex[1] that is large enough not to alter the edges of the convex hull of the original set. On the other hand, it must not be too large, because otherwise it could make the construction of the triangulation numerically unstable. The correct size of these simplices has been adressed by [34, 56] and experiments show that the ideal way to construct the initial simplex is the one illustrated in Fig. 3.1.

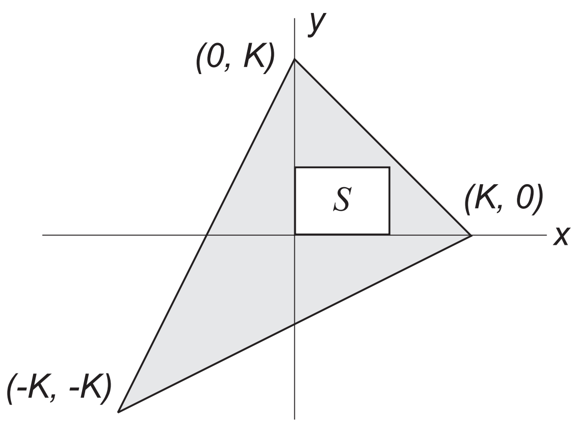


Figure 3.1: The ideal initial simplex construction for the incremental insertion algorithm.

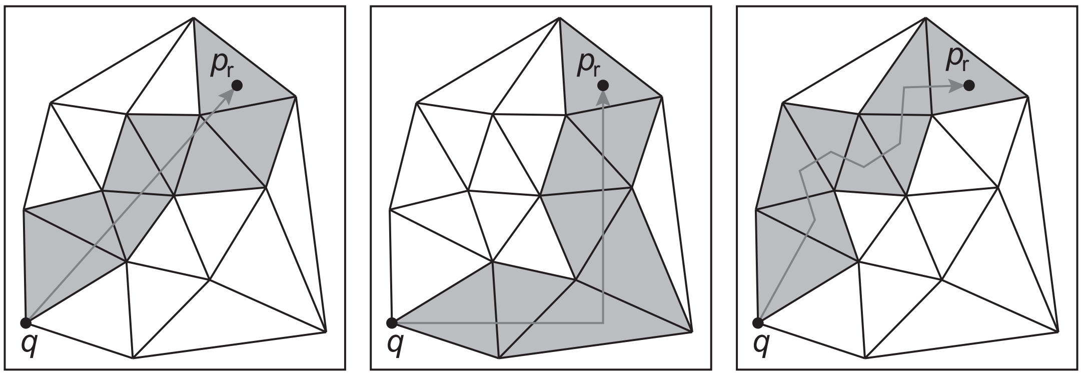
[1]This simplex will be a triangle E^2, a tetrahedron in E^3, etc.

As we can see in the figure, the ideal simplex in E^2 should be constructed using the vertices with coordinates $[K, 0]$, $[0, K]$ and $[-K, -K]$ where K is equal to approximately $\kappa \cdot \max(x_{max}, y_{max})$, where x_{max} and y_{max} are the width and the height of the bounding box of S as shown in Fig. 3.1 and κ is a real constant. The referenced sources show that $\kappa = 10$ or $\kappa = 3.5$ are good choices but other simillar numbers should work as well.

3.3.3 Point Location

As shown in Alg. 3.1, in order to add a point into the triangulation, the triangle containing its location has to be found. Generally, there are two different classes of approaches for point location – the approaches based on the walking algorithms and the approaches based on special location data structures.

The walking algorithms take advantage of the fact that no special data structure is needed. Each point location process starts in an arbitrary triangle and by following a given set of conditions traverses the triangulation until the triangle containing the given point p_r is reached. According to [48, 15], there are three main types of the walking algorithms, based on the type of traversal they use – see Fig. 3.2 (all the walks start in the triangle containing the point q).



(a) Straight walk (b) Orthogonal walk (c) Stochastic walk

Figure 3.2: Three main types of walking algorithms. Images courtesy of Jiří Skála.

Straight walk: Each triangle is traversed, which lies on the line segment connecting the starting triangle and the given point p_r.

Orthogonal walk: Two axis-aligned line segments are created that connect the starting triangle with the given point p_r. The triangulation is then traversed along these two line segments. According to [48], the traversal may not end in the correct triangle and it is necessary to combine it with some other traversal type.

Stochastic walk: The next triangle of the stochastic walk algorithms traversal is chosen by randomly picking an edge of the currently processed triangle and testing if the target point p_r lies on the other half-plane given by this edge than the rest of the current triangle (an orientation test is usually used as shown in Eq. 2.5). If it does, the algorithm traverses to the neighbouring triangle by walking over the chosen edge. In the other case, the next edge of the current triangle is tested and so on. After three failed tests, we know that the traversal has reached the triangle containing p_r (or more precisely, we know that we have reached the target triangle after two failed tests since we do not usually test the edge common with the previously visited triangle).

As shown in [38, 48] and others, the time needed by the walking algorithms to locate a point strongly depends on the selection of the starting triangle. Depending on the choice of the starting triangle, the expected time complexity of a walk algorithm may vary from $O(n^{1/2})$ to $O(n^{1/4})$ which is worse than the optimal $O(\log n)$.

The other approach for triangle location uses special auxiliary data structures such as Directed Acyclic Graphs (DAG), skip-lists and others (see [10, 34, 14]). These special data structures usually work by creating a hierarchy of the triangles in the triangulation. Let us have a look at the DAG structure – as we can see in Fig. 3.3, we start with a simple triangulation. The associated DAG structure starts with a pointer to each of its triangles. As new points are being added into the triangulation and the edges are being flipped in order to maintain the Delaunay property of the triangulation, we may see that two things happen in the DAG. When a triangle is subdivided into three triangles ($T_2 \rightarrow \{T_4, T_5, T_6\}$) a new level in the DAG is created and the original triangle is connected to its children with pointers. When an edge flip occurs, all the upper level triangles that contain at least part of the newly created triangles are connected to them via new pointers. As we can easily see, the DAG structure is organized in a tree-like fashion and thus enables point location with expected time complexity of $O(\log n)$.

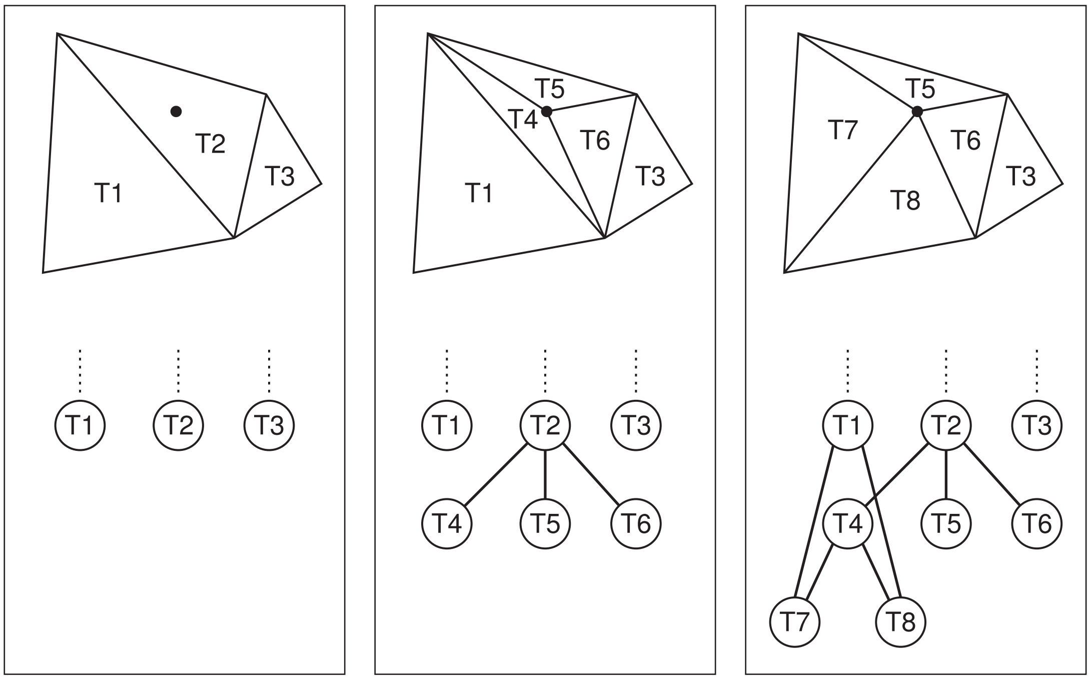
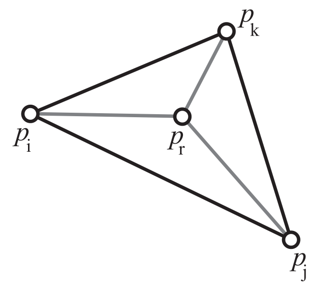


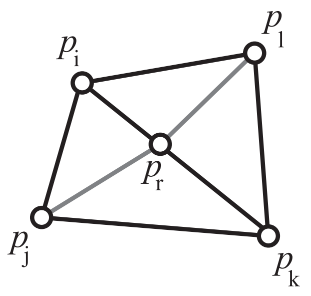
Figure 3.3: A simple triangulation and an associated DAG. Image courtesy of Jiří Skála.

3.3.4 Point Insertion and Edge Legalization

Once the triangle $p_i p_j p_k$ containing the given point p_r is located, two cases may occur (considering two dimensional triangulation) – p_r may either lie inside $p_i p_j p_k$ or on one of its edges. Theoretically, a third special case might occur when p_r is identical to one of the vertices of the target triangle, but only if we allow the set of input data S contain multiple identical points (as shown in Alg. 3.1, we require S to contain n *distincts* points). The process of handling both the allowed cases is similar – see Fig. 3.4.



(a) p_r is added inside a triangle



(b) p_r is added on an existing edge

Figure 3.4: Triangle splitting after adding a new point into the triangulation.

In Fig. 3.4(a), we can see the case when p_r is added into an existing triangle $p_i p_j p_k$, dividing it into three new triangles. Fig. 3.4(b) shows the other possible case, when p_r is added on the edge common to two triangles – $p_i p_j p_k$ and $p_i p_k p_l$, these are then divided into four new triangles.

Because of the fact that new triangles are being added into the triangulation without paying respect to the Delaunay condition which has to be preserved, it is necessary to check if the newly created triangles satisfy this condition. If the newly added triangles do violate the Delaunay condition, they have to be flipped (let us call these edges *illegal*):

Definition 3.3.1 (Edge flip)**.** Given two adjacent triangles $p_i p_j p_k$ and $p_j p_k p_l$, an edge flip performed on these triangles replaces them with triangles $p_i p_j p_l$ and $p_i p_l p_k$.

After the illegal edges are flipped, the legality test is recursively performed on the outer edges of the new triangles, created by the flipping. This process is shown in Alg. 3.2.

```
Input:
  • $p_r$ – a point being inserted into $T(S)$
  • $p_i p_j$ – the edge of $T(S)$ that may need to be flipped
  • $T(S)$ – a triangulation
Output: $DT(S)$ – Delaunay triangulation
if $p_i p_j$ is illegal then
    Let $p_i p_j p_k$ be the triangle adjacent to $p_r p_i p_j$ along $p_i p_j$
    Replace $p_i p_j$ with $p_r p_k$                                  // Flip $p_i p_j$

    LegalizeEdge($p_r$, $p_i p_k$, $T(S)$)
    LegalizeEdge($p_r$, $p_k p_j$, $T(S)$)
end
Return $DT(S)$                                                  // $T(S) \equiv DT(S)$
```

Algorithm 3.2: Edge legalization in the process of $DT(S)$ construction.

3.4 Summary

The incremental insertion algorithm for Delaunay triangulation construction is online (as we can easily add new points later, they do not necessarily have to be in the initial generator set), it may be parallelized and extended to three dimensions. Its time and space complexity depend solely on the used method of point localization. Even although the special data structures enable us to run this algorithm in optimal time complexity, it is often more convenient to use the walk-based approach, since it is much easier to implement and maintain and its performance is usually good enough.

Due to its properties, the incremental insertion algorithm using a walk-based point location represents the ideal choice for managing the kinetic dataset. This is caused by the facts, that no auxiliary data structure is needed, the runtime is nearly optimal and we are able to add and remove points from the triangulation relatively easily. The simplicity of the structure of a triangulation created by this algorithm makes it extremely easy to handle and update as its topology changes due to the point movement, the online property on the other hand allows us to manipulate the dataset at any moment during the lifecycle of the application and (together with some point removal algorithm such as the one presented in [13]) makes the triangulation extremely versatile and even allows us to simulate the movement by the dynammic approach. Finally, the non-optimal construction time is of a relatively little concern to us, since the construction step is usually performed as a preprocessing and does not influence the movement of the points.

Chapter 4

Handling the Kinetic Datasets

This chapter describes the kinetic data structures with a focus on kinetic Delaunay triangulation. The problem of managing a planar Delaunay triangulation of a set of moving points is discussed in detail as it represents the main area of our research.

At first, the difference between the dynamic and the kinetic approach to point movement is shown. After that, general functionality of the kinetic data structures (KDS) is presented with special respect to the kinetic Delaunay triangulation (KDT). Furthermore, the kinetic Delaunay triangulation is described in greater detail – the principle of kinetization of this data structure is shown and most commonly used methods for polynomial root finding are shown as the polynomial functions play a key role in the process of kinetization. Finally, general properties, which are most commonly used for kinetic data structures eveluation, are described and used for the evaluation of the kinetic Delaunay triangulation.

4.1 Kinetic & Dynamic Approach

In general, there are two different approaches for handling movement in the given dataset (in our case a set of points). The movement may be discretized and simulated by removing the points from the given data structure and reinserting them back at new possitions as if they moved there or it may be understood as a continuous change of the dataset. To distinguish between those two approaches, the discrete one is often referred to as the dynamic movement whereas the continuous one is often considered to be the true kinetic approach (first introduced in [5]).

Dynamic approach: there are several advantages of the dynamic approach – first of all, its performance is completely independent on the type of the movement of the points. The points may move along any type of trajectory with no restrictions on acceleration or other movement properties. The runtime consumption is then dependent on the algorithms for point insertion and removal. On the other hand, the discretization of movement generates some drawbacks, for instance it cannot be reliably used for collision detection and similar problems; [37] shows that this type of data manipulation is most commonly used for simulating time-dependent datasets where some points are only used for a limited time period. Generally speaking, we may say that the dynamic approach to movement simulation tends to suffer from two problems – either it is oversampled and we are wasting computational resources on computing unnecessary data, or it is undersampled and we miss important events.

Kinetic approach: the over/under-sampling difficulties of the dynamic approach may be overcome by using the kinetic approach – by computing a certain time events, we are able to determine when the underlying data structure changes and needs our attention (topologic update, etc.).

This type of simulation is then obviously strongly dependent on the type of the underlying data structure, the type of movement as these two properties determine the type of computation that needs to be performed in order to keep its properties. However, the data can be accessed at any given time without any loss of information. Furthermore, according to [25, 46], in certain applications, it is faster to simulate the movement using this type of approach.

4.2 Kinetic Data Structure Cornerstones

In order to be able to kinetize an ordinary (i.e., static) data structure, we have to understand the basic cornerstones of the kinetization process. These will be described in this section. Although the described properties are general and may be (and they often are) appplied to any type of different kinetic data structures, we will only use the special case of kinetic Delaunay triangulation as an example, because it is the only kinetic data structure within the scope of our work and it has also been described in the previous text in its static version.

4.2.1 Predicates and Certificates

As shown in Chapter 2, the spatial division data structures are often defined by special functions called predicates. Examples of such predicates include the incircle and orientation test as shown in Eq. 2.4 and Eq. 2.5 respectively. As shown in [5, 25], each geometric structure constructed over a set of geometric primitives may be proved valid by checking a finite numbers of predicates of these primitives. These checks are then called *certificates*. In the case of Delaunay triangulation, the certificates are represented by the incircle test functions (Eq. 2.6) because we are able to determine if any given triangulation $T(S)$ is Delaunay by performing the incicrcle test on each pair of adjacent triangles in the triangulation (if S is finite set, then the triangulation will have a finite number of edges and thus we will have to perform finite number of tests).

4.2.2 Point Movement Description

Since we want to manage a Delaunay triangulation over a set of moving points, it is necessary to define the point movement. Let us say that moving point is such a point which has time-dependent coordinates:

$$p_i(t) = (x_i(t), y_i(t)) \tag{4.1}$$

where $t \in \mathbb{R}$ represents the time variable. For the purposes of physics-based applications, it is most commonly reasonable to use only continuous functions of time for the point coordinates. Furthermore, in the field of kinetic data structures, only polynomial functions are usually used for the coordinates.

Theoretically, it is possible to use other functions than polynomials for describing the point movement, but only polynomials are practically used. This is caused by the fact that they provide us with relatively wide variety of options of the point behavior and are relatively easy to compute (see further). The polynomials also represent one of the best choices because we usually require the given functions to behave in an algebraic (or at least in *pseudo*-algebraic) fashion, which means that we require the functions to have a finite number of roots (see [26]). Furthermore, the polynomials (albeit of high degrees) may be used to interpolate almost any physically tangible trajectory, for instance by using the Taylor expansions (see [1]).

4.2.3 Certificate Functions

If we allow the points to move continuously by replacing their coordinates by functions of time, the certificates themselves will become functions of time – *certificate functions*. The type of these functions depends both on the original certificates and on the type of point movement we allow. As we have already mentioned in Chapter 2, the certificates are most often functions of the input data and are commonly in the form of determinant of a matrix (such as the incircle test function in the case of the Delaunay triangulation – see Eq. 2.4). In this case, we have to compute a determinant of a 4×4 matrix in order to determine if a point lies inside, outside or on a circumcircle of any given triangle (considering a planar triangulation). If we replace the static points by moving points as defined in Eq. 4.1, we obtain a time dependent matrix incircle test in the following form:

$$\det \mathbf{I}(t) = \det \begin{bmatrix} x_i(t) & y_i(t) & x_i^2(t) + y_i^2(t) & 1 \\ x_j(t) & y_j(t) & x_j^2(t) + y_j^2(t) & 1 \\ x_k(t) & y_k(t) & x_k^2(t) + y_k^2(t) & 1 \\ x_l(t) & y_l(t) & x_l^2(t) + y_l^2(t) & 1 \end{bmatrix} \tag{4.2}$$

where (similarly to the static case) we may compute the position of $p_l(t)$ against a circumcircle of triangle $p_i(t)p_j(t)p_k(t)$ for any given time $t \in \mathbb{R}$. Furthermore, we may use the certificate functions to determine *if* and *when* the certificate will cease to be valid (for instance when $p_l(t)$ enters the circumcircle of $p_i(t)p_j(t)p_k(t)$) – see further.

As we can see, if polynomial functions are used as the coordinate functions, the certificate function becomes a polynomial function itself. Especially, if only linear functions are used for the coordinates, the certificate function becomes a polynomial of the fourth or lower degree (in this case, the points move along linear trajectories without any acceleration). Certificate functions are quite common for various types of kinetic data structures.

4.2.4 Kinetic Events

As we have said, with the change of the time, the value of the certificate functions also changes. As long as its sign remains unchanged, the certificate remains valid and the kinetic data structure remains unchanged (apart from the point movement). It is important to note that the movement of the points does not necessarily means the need to perform changes in the kinetic data structure. Let us for instance consider a situation when all the points in $DT(S)$ move in the same direction with the same velocity – the whole triangulation would change its position but it would remain unchanged topologically.

On the other hand, in the common situations, instants in time will usually occur when the sign of the certificate function changes. These instants indicate that the structure has lost its properties and in order to preserve its validity, we have to perform a change in it (such as the case of a point entering the circumcircle as we mentioned above). These instants are called *kinetic events* because they are caused by the movement of the points, or certificate failure times because they denote that the certificate function has failed. In the particural case of kinetic Delaunay triangulation, we call them *topologic* *events* because a change in the topology of the triangulation has to be made – in order to retain the Delaunay property, we have to perform an edge flip as defined in Def. 3.3.1.

Internal and External Kinetic Events

Two types of kinetic events are usually described in the literature (see [5, 26]) – the *internal* and the *external* kinetic events. The difference between these two types of events is that the external events (as defined in [26]) are those which directly change the configuration of the kinetic data structure. An example of an external topologic event is the already mentioned edge flip in the kinetic Delaunay triangulation – it needs to be done in order to retain the Delaunay property of the triangulation and it changes its topology. The internal events only exist in some kinetic data structures (we do not find them in kinetic Delaunay triangulations) and they are not ”visible from the outside”, they are such events that need to be processed for the KDS to work correctly but do not directly change its structure. They may be found for instance in the structures for managing closest pair relations on kinetic data sets (see [5]).

Event Queue

In order to be able to manage the kinetic structure continuously, it is vital to process the kinetic events in the correct order. If two different certificates fail (i.e., the corresponding certificate functions change signs), it is necessary to process the according changes in the kinetic data structures in the correct order. A priority queue is most commonly used for this task (see [5]) – the certificate failures are stored in the queue with the priority being the time when they will occur. The lifecycle of a kinetic data structure then usually consists of storing some events in the queue and then periodically popping some of them from the queue, computing new events, pushing them into the queue and so forth as needed – see further.

4.3 Kinetic Delaunay Triangulation

This section will provide an overview of the kinetization of the Delaunay triangulation as it is the data structure which is in the focus of my research.

4.3.1 Kinetization Principle

Because of the locality of the Delaunay condition (already mentioned in Section 2.2.1), we know that if a certificate becomes invalid, it is sufficient to flip the edge common to the two triangles involved in the certificate, deschedule the events that included the removed triangles and reschedule new events for the newly created triangles and their neighbours. There is only one type of certificates needed for proving that a given triangulation is Delaunay – the incircle test already shown in Eq. 2.4 (which is performed on a pair of adjacent triangles) and thus only one type of certificate functions needs to be evaluated. The whole process of KDT management is shown in Alg. 4.1.

As we may see the lifecycle of KDT is composed of two steps. In the first step (which is called the initialization step), we have to compute the first event that will occur for each pair of adjacent

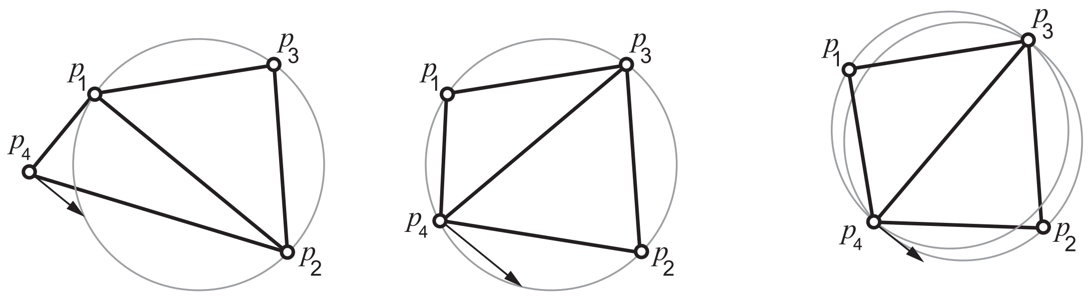
```
Input: $DT(P)$ – Delaunay triangulation of kinetic data
Auxiliary:
  • $t_{curr}$ – The current time of the triangulation
  • $Q$ – Priority queue

// Initialization step
foreach Adjacent triangle pair $T_i$, $T_j$ in $DT(P)$ do
    Try to compute the next future topological event $\Sigma_{ij}$ at time $t_{ij}$
    if $\Sigma_{ij}$ exists then
        $Q.push(\Sigma_{ij}, t_{ij})$
    end
end
// Iteration step
while Time of $Q.head() < t_{curr}$ do
    $\Sigma \leftarrow Q.pop()$
    Handle $\Sigma$
    Push new events into $Q$ as required
end
/* This step is repeated as required during the whole lifecycle for
   increasing values of $t_{curr}$                                    */
```

Algorithm 4.1: Lifecycle of a kinetic Delaunay triangulation.

triangles in the whole triangulation. These events are then stored in a priority queue, where they are ordered according to the time when they will occur. Since the iteration step has to be performed for each pair of adjacent triangles in the triangulation, we may see that the complexity of this step is $O(3\|S\| + 2\|C\| - 3)$ where S is the set of generating points and $C \subset S$ are the generators located on the convex hull of S.

The second step (called the iteration step) consists of repeated updating the time of the triangulation according to our needs (e.g., obtaining a single state for each frame during a rendering loop) and testing if an event occured between the last state and the wanted one. If there is an event on the head of the event priority queue, we pop it from the queue, handle it by flipping the common edge of the relevant triangle pair and recompute several new events which are then pushed into the queue. This step is repeated as long as there is an event at the head of the queue which takes place between the last valid state and the currently wanted one. The number of repeating the iteration step depends on several variables – the type of the point movement, the total amount and the location of the points in the triangulation (both moving and static points), etc. It has been adressed by many authors and represents a non-trivial problem – see [3].

An illustration of event handling in KDT is given in Fig. 4.1. In Fig. 4.1(a), we can see a point marked as p_4 moving inside a static triangle $p_1p_2p_3$ (none of its points is moving). As p_4 reaches the circumcircle of $p_1p_2p_3$ (shown in Fig. 4.1(b)), the assigned certificate function reaches a zero value which causes the certificate to fail. In order to keep the Delaunay property, we have to perform an edge flip (which is done by replacing the edge p_1p_2 by the edge p_3p_4, as we can see in Fig. 4.1(c).



(a) Initial situation (b) p_4 enters the circumcircle of $p_1p_2p_3$ (c) Edge flip is performed

Figure 4.1: Topologic event in the Delaunay triangulation.

It is important to say that not every scheduled event will be executed. Let us have a look at the example shown in Fig. 4.2. The situation shown in this figure is similar to the previous case – there is a static triangle $p_1p_2p_3$ and a point p_4 moving into its circumcircle. On top of that, there is also a point p_5 moving towards the circumcircle of triangle of $p_2p_4p_3$. The topologic events that are likely to occur with this point configuration are marked by the times when the points enter the circumcircles ($t_0 < t_1$). Let us then compute both of these events and push them into the priority queue. When the triangulation reaches time $t = t_0$, event over points p_2, p_3, p_4, p_5 is handled by flipping the edge p_3p_4 for p_2p_5. At this moment, the triangle $p_2p_4p_3$ ceases to exist, making the event scheduled for $t = t_1$ invalid. Such an event must be dequeued and the time spent on its computation is lost.

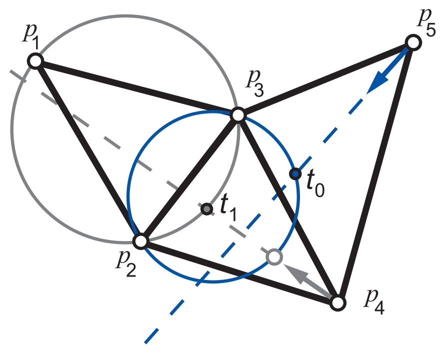


Figure 4.2: Redundant topologic event.

4.3.2 Event Computation – Mathematical Preliminaries

As stated before, for several reasons, it is often convenient to restrict the allowed point movement to polynomial functions of time, let us also follow this restriction by defining the moving points as shown in Eq. 4.1 – i.e., with the point coordinates $x_i(t), y_i(t)$ being polynomial functions of variable $t \in \mathbb{R}$. In this case, the certificate function becomes polynomial of degree no greater than $4 \cdot \max_i\{\deg p_i\}$. Particullary for linear trajectories of the points, the certificate function will become a fourth or lesser degree polynomial. Let us now have the certificate functions in the form of a polynomial:

$$c(t) = \det \mathbf{I}(t) = \sum_{i=0}^{n} a_i \cdot t^i \tag{4.3}$$

where $\mathbf{I}(t)$ is the certificate function for KDT as shown in Eq. 4.2, $n \leq 4 \cdot \max_i\{\deg p_i\}$ is the degree of the resulting polynomial, $t \in \mathbb{R}$ is the time variable and $a_0, \dots, a_n$ are the coefficients of the polynomial.

Before we describe the most commonly used methods for polynomial root location, let us first introduce some bounds on their position (note that we are only interested in the real roots of each certificate function):

Root Position Bounds

Given a polynomial equation as shown in Eq. 4.3, according to [30] the value of each of its real roots z_i may be restricted as follows:

$$|z_i| = \max\left\{1, \frac{1}{a_n}\sum_{i=0}^{n}|a_i|\right\} \tag{4.4}$$

In other words, we may say that for each polynomial, there is an interval that contains all of its real roots. The size of this interval may be computed by using solely the coefficients $a_0, \dots, a_n$ of this polynomial.

Descartes' Rule of Sign

This simple rule (discussed for instance in [45]) allows us to simply obtain an upper bound on the number of the real roots of a polynomial on any given interval:

Theorem 4.3.1 (Descartes' Rule of Sign)**.** *Let us have a polynomial $c(t)$ as in Eq. 4.3. If we denote $V(c)$ the number of sign variations in the sequence $(a_0, \dots, a_n)$, discarding all zero values, and $||c_+||$ the number of all the positive roots of $c(t)$ with multiplicities, then $||c_+|| \leq V(c)$ and $V(c) - ||c_+||$ is even.*

Using the transformations of paramater t as shown in [45] will allow us to determine the number of real roots at any given interval.

Sturm Sequences of Polynomials

Definition 4.3.1 (Sturm Sequences, [44])**.** The sequence of polynomials

$$f_1(x), f_2(x), ..., f_m(x)$$

will be Sturm sequence at interval $\langle a, b\rangle$ (a and b may be infinite), if:

1. $f_m(x)$ is nonzero at the whole interval $\langle a, b\rangle$
2. The two adjacent polynomials to the polynomial $f_k(x), k = 2, ..., m-1$ are nonzero at zero points of this polynomial and have the opposite signs there, thus:

$$f_{k-1}(x) f_{k+1}(x) < 0$$

The key feature of the Sturm sequences of polynomials is that they may be used to compute the total number of polynomial roots on any given interval including their multiplicities. Consequence of this fact is that Sturm sequences may be quite easily used for polynomial root computation. Sturm sequence of a polynomial $f(x)$ may be constructed (as shown in [44]):

$$\begin{aligned} f_1(x) &= f(x) \\ f_2(x) &= f'(x) \\ f_{j-1}(x) &= q_{j-1}(x)f_j(x) - f_{j+1}(x), j = 2, ..., m-1 \\ f_{m-1}(x) &= q_{m-1}(x)f_m(x) \end{aligned} \tag{4.5}$$

In these relations, $q_{j-1}(x)$ is the quotient and $f_{j+1}(x)$ is the negation of the remainder of division of the polynomial $f_{j-1}(x)$ by the polynomial $f_j(x)$. $\{f_i(x)\}$ is thus a sequence of polynomials of a decreasing degree. The first term of the sequence is the input polynomial, the second term is its derivate and each of the following terms $f_i(x)$ is obtained by computing the remainder of the division $\frac{f_{i-1}}{f_{i-2}}$ and changing the sign of this remainder.

These facts may become easier to observe for the reader, if we rewrite the third equation from Eqs. 4.5 to the following form:

$$\frac{f_{j-1}(x)}{f_j(x)} = q_{j-1}(x) + (-1) \cdot \frac{f_{j+1}(x)}{f_j(x)} \tag{4.6}$$

What we see in Eq. 4.6 is a division of two polynomials, with $f_{j-1}(x)$ being the numerator and $f_j(x)$ being the denominator of the division. $q_{j-1}(x)$ then denotes the quotient (which is not used for the practical computation of the Sturm sequences) and $f_{j+1}(x)$ is the negation of the remainder of the division (the multiplication of $f_{j+1}(x)$ by the constant -1 is necessary to make the relation mathematically correct.

Let us now define the function $V(x)$ as the count of the number sign changes in the Sturm sequence 4.5 (ignoring all zeroes). This function may then be used to count the number of distinct real roots $r_{\langle a,b\rangle}$ of $f(x)$ on any interval $\langle a, b\rangle$:

$$r_{\langle a,b\rangle} = V(a) - V(b) \tag{4.7}$$

where $a, b \in \mathbb{R}$ or either of a, b may be infinite. As proved in [44], Eq. 4.7 remains valid even if a or b are the roots of $f(x)$. Furthermore, the last term of the Sturm sequence 4.5 may be used to distinguish and compute the values of the multiple roots of $f(x)$. All the multiple roots of $f(x)$ with multiplicities decreased by one are the roots of $f_m(x)$, which does not have any other roots. Together with the fundamental theorem of algebra, this statement may be extended to various useful conclusions. For instance if $f_m(x)$ is of an odd degree, then $f(x)$ has at least one multiple root, etc.

Generalized Sturm Sequence

Note that, if the initial polynomial has some multiple roots, the created sequence is no further a Sturm sequence as defined in Def. 4.3.1, because the second required condition is not met. In this case, the sequence is called a generalized Sturm sequence and has all the aforementioned features. The generalized Sturm sequence is formally defined as an extension of Sturm sequence $\{f_i(x)\}$ by multiplying all of its terms by any polynomial $p(x)$, thus gaining a sequence in the form of $\{p(x)\cdot f_i(x)\}$. If a Sturm sequence is mentioned anywhere in the following text, a generalized Sturm sequence is meant.

4.3.3 Event Computation – Polynomial Solving Methods

This section provides an overview of the most commonly used methods for polynomial root location in the context of the kinetic Delaunay triangulations. According to [27] these methods may be divided into three basic categories:

Naive approach: the most straighforward method consists of computing all the real roots of a given polynomial, discarding the roots that are not usable (e.g., all the negative roots) and enqueuing new events that occur at the times determined by the remaining roots (if there are any left). The methods used for computing these roots may include such approaches as *Laguerre method*, *eigenvalue based solvers*, *Newton method*, *Bairstow method* and other numerical methods or their combination. Details on variety of these methods may be found in literature – see [18, 42, 44] and others.

Hybrid methods: using the preliminaries we have shown earlier (such as the Descartes' rule of sign and the Sturm sequences of polynomials), these methods are able to divide the real axis into intervals containing the individual roots of a given polynomial. Using some iterative approach (such as the Newton method or bisection), we are then able to determine the location of the roots. In order to be able to employ these iterative methods efficiently, the polynomials may be replaced by other polynomials with exactly the same roots but with multiplicities equal to one (see [27]). Details on these methods may be found in [27, 45, 46].

Interval based methods: first proposed in [27], these methods represent a different approach. Instead of computing the polynomial roots, we rather use intervals that contain only one root each. The intervals may be of virtually any size when they are computed and they are reduced when the need to do so emerges – for instance if we need to store them in the priority queue or determine which root certificate should be handled first. The main advantage of this approach is the fact that not each root becomes computed exactly and thus we can save runtime on not computing the roots that we do not really need (the events may become de-scheduled as we explained earlier). Details on this approach may be found in [27, 46].

Guibas and Karavelas also provide a comparison of some representants of the aforementioned methods in [27], where they show that both the hybrid and the interval based methods are better than the numerical solvers (in the terms of runtime consumption) for polynomial trajectories, where the certificate function result in a polynomial of degree greater than six. For the lower degree polynomials, the numerical methods are quicker.

Note. Let us note that analytical methods for polynomial solving are not practically usable as they only allow us to solve polynomials of low degrees and they usually need to use complex numbers even in the process of finding real roots, which is considered to be a potential source of errors.

4.4 Kinetic Data Structures Properties

According to [5, 6, 26], there are four basic properties that may be used to judge the effectiveness of a particular data structure for kinetic data. These are *responsiveness*, *efficiency*, *locality* and *compactness*. Note that these properties are dependent on the exact implementation of the given kinetic data structure, not only on its type and the property it is maintaining. For the purposes of the KDS properties description, let us denote that a given KDS contains n moving primitives.

Responsiveness: Responsible kinetic data structure is such a KDS that takes only a small amount of time to handle each certificate failure (i.e., a kinetic event). Handling of the event may include determining a new configuration of the data structure, altering its properties, adding new certificate functions or removing old ones (as the structure may change), etc. If expressed as a function of n, a KDS is responsible if its responsibility is small – i.e., of order $O(\log^{\epsilon}(n))$ or $O(n^{\epsilon})$ for an arbitrarily small ϵ.

(Weak) Efficiency: A kinetic data structure is said to be efficient if the total number of events processed in the worst case is asymptotically the same as (or slightly larger than) the number of external events processed in the worst case. Let us note that these two ”worst cases” do not necessarily need to be the same cases (which is the reason that this property is sometimes called *weak* efficiency).

Strong Efficiency: The difference between strong and weak efficiency is that the KDS is strongly efficient if and only if the worst case ratio between totally processed events to processed external events is small for any given motion.

Locality: As a small change in the kinetic data structure may influence quite a large number of certificates, it is often convenient to be able to determine the relationship between each primitive and the certificates it may directly affect. A kinetic data structure is called local if the number of directly affected certificates is small for any input data.

Compactness: As said before, the kinetic data structures use a priority queue for event management. The length of the queue may be then expressed as a function of the size of the input data n. A given data structure is then compact if the maximum number of events ever present in the queue is small, i.e., (using the same notation as above) is of order $O(\log^{\epsilon}(n))$ or $O(n^{\epsilon})$ for an arbitrarily small ϵ, where n is the size of the set of the input data.

4.4.1 KDT Evaluation Using KDS Properties

Given the kinetic data structures properties, we may use them to evaluate the KDT (similar evaluations may be found in [26, 6], where the properties of KDT and other kinetic data structures are explored). Using the knowledge from the previous text, we may state that KDT has the following properties:

- KDT is *responsive* – managing an event takes almost constant time for any event.
- KDT is *strongly efficient* – only one type of events exist, and thus all the events are external. The ratio of all processed events to processed external events is $O(1)$.
- KDT is not *local* – each moving point may be connected to a relatively large number of neighbours and thus participate in a relatively large number of events.
- KDT is *compact* – although the precise relations between the size of the input set, the allowed type of movement and the size of the event queue are not known, the compactness of KDT has been shown – see [3, 26].

4.5 Summary

The kinetic data structures and especially the kinetic Delaunay triangulation have been undergoing an extensive research in the past years. Result of this is that they are relatively very well understood, especially in their planar variants (there is a research still going on in higher dimensions, see [7, 36, 46]). One of the main issues in this field (and perhaps the most important one) is the way of certificate failures (i.e., events) computation. Various methods are presented with the modern trend being the use of interval approach (see [46]) or a speed-optimized numerical solvers (as in [7]). Each of these methods is usable in different conditions – the numerical solvers are usually quicker for polynomials of low degree, whereas the polynomials of degree greater than six are better solved by the interval based methods. We may see that this feature also partially determines the field of use of these methods. We may expect to see the numerical solvers in the applications where the continuity of the movement does not play a key role and we may simulate curved trajectories by piecewise linear curves (we already mentioned that points moving along linear trajectories provide us with low order polynomials as certificate functions). On the other hand, if we want the movement to be smooth, we will most probably use a more sophisticated method which will ensure better runtime consumption for higher order polynomials.

Chapter 5

Applications of Kinetic Data Structures

This chapter will provide the description of some of the applications that benefit from using the kinetic data structures (namely the kinetic Delaunay triangulation and similar structures).

5.1 Collision Detection

Gavrilova proposed in [22] the concept of using the kinetic Delaunay triangulation as a means of collision detection between points in the generator set. The principle of this method lies in the fact that only such points may collide that share a common edge in the Delaunay triangulation. The principle that allows us to use the kinetic Delaunay triangulation lies in its duality with the Voronoi diagram. Two colliding points are in fact a limit example of extremely close points. And as we have already shown in Eq. 2.1, the closer two points are, the more probable it is that their cells will neighbour each other in the Voronoi diagram. Together with the principle of duality, we may state that there will be a Delaunay edge connecting the two colliding points. Because all the points in the generator set may be moving, the edge between two colliding points will be present in the triangulation unspecifiably earlier than the collision occurs.

Erickson et al. shown in [19] that the kinetic approach may be used to detect collisions between convex polygons in E^2. This work has later been extended by Guibas et al. in [28]. The later work then uses a kinetic regular triangulation, where the point weights are used to represent the radii of bounding spheres of the moving polyhedra.

Some practical applications of kinetic collision detection may include such applications as the one proposed by Goralski and Gold in [24] where the kinetic Voronoi diagrams are used to provide the spatial relations as an aid for human navigators of marine vessels – see Fig. 5.1. They may then use the provided information to improve their performance, as the main cause of accidents in this field is, according to the referred paper, the human error.

Another practical use of the kinetic data structures (a kinetic regular triangulation in this specific case) is the work of Ferrez, see [20], where the kinetic regular triangulation is used to simulate movement of a granular material (which incorporates the collision detection among the grains). Similarly as in the already mentioned work of Guibas et al. (see [24], the weights of the points in the simulation are used to represent the radii of the individual grains.

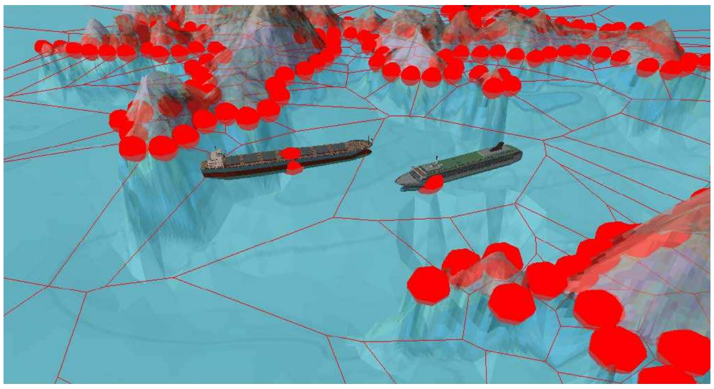


Figure 5.1: An example of the collision detection application of kinetic Voronoi diagram in the marine environment, [24]

5.2 Simulation of Crowds

As presented by various research groups, the problem of crowd simulation[1] may be handled in many different ways which are usually based on the adaptations of one of the basic approaches which include agent based systems (examples may be found in [23, 33, 41, 49]) and potential based approach (such as the one presented in [29, 50]).

If we choose to use an agent based simulation (each pedestrian is considered to be an autonomous entity with possibly unique behaviour), we usually face two different problems. At first, there is the problem of global navigation as it is necessary to navigate the pedestrians through the virtual environment and each of the pedestrians may have a unique point of origin and also a unique target. As shown in the referred sources, this problem is usually handled by using some kind of graph structure that covers the whole environment and enables the global navigation. The other problem covers the local behaviour of the pedestrians and basically consists of collision avoidance (as it is unlikely for two pedestrians to collide in real situations) and of local behavior management such as group coherence preservation.

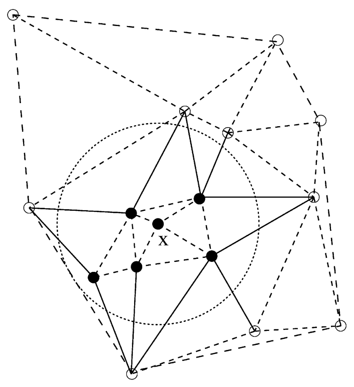


Figure 5.2: Crowd agent with a safe zone and connection to other nearby agents, [23].

[1]Let us only consider the simulation of human crowds but note that the simulations of flocks of birds, schools of fishes and similar phenomena falls into the same category and is commonly handled by using very similar approaches.

Goldenstein et al. shows in [23] that the local behaviour of the pedestrians may be handled by using a special kinetic data structure, which is in fact a special case of a kinetic Delaunay triangulation. In this approach, each agent is assigned a ”safe zone” and the kinetic data structure is used to keep track of the other agents that are nearby and may enter this safe zone. As we may see in Fig. 5.2, each agent (marked with the X in this figure) considers not only its nearest neighbours, but also some of their neighbours and so on if the concentration of agents is localy dense. In order to do that, it is necessary to add a new type of events to keep track of other agents entering or exiting the safety zone (agents inside the safety zone of X are shown as black circles).

5.3 Mathematical Simulations

The simulations of various different natural phenomena, such as fluid dynamics, requires to solve systems of partial differential equations. In practise, these equations are usually discretized and solved using finite difference method (*FDM*) or finite element method (*FEM*) (these methods will not be discussed because they are out of the scope of this work). Beni addresses the problem of PDE solving in [7] in the field of Geographic Information System (GIS) simulations and suggests that the kinetic data structures (namely the kinetic Delaunay triangulation and kinetic Voronoi diagram) may be used for the purposes of fluid dynamic simulations in E^3.

The Voronoi diagram is shown to be very well usable for the purposes of fluid representation as it is dynamic (incremental insertion algorithm is used for its construction) and thus allows mesh refinement in areas that need to be more detailed. Furthermore, the Voronoi cells are well-shaped and therefore provide good basis for the differential equations. To prove the usability of this approach, three different case studies are also presented in [7] – reflection of a compressible fluid inside a solid bounding cube where it is shown that the particles will compress upon reaching the walls of the bounding area, their velocity drops and their temperature, density and presure increases. The second case study shows one-directional expansion of gas in a tube; in this case the particles with higher density (which are in the middle of the tube) tend to move outwards and their volume increases. The last case study shows a fully three dimensional expansion of gas – the particles are initially pressurized in the centre of the simulation area and then begin to move outwards (see Fig. 5.3). Results from these case studies show that the kinetic Voronoi diagrams represent a valid choice for fluid dynamics simulations and are comparable with other commonly used approaches.

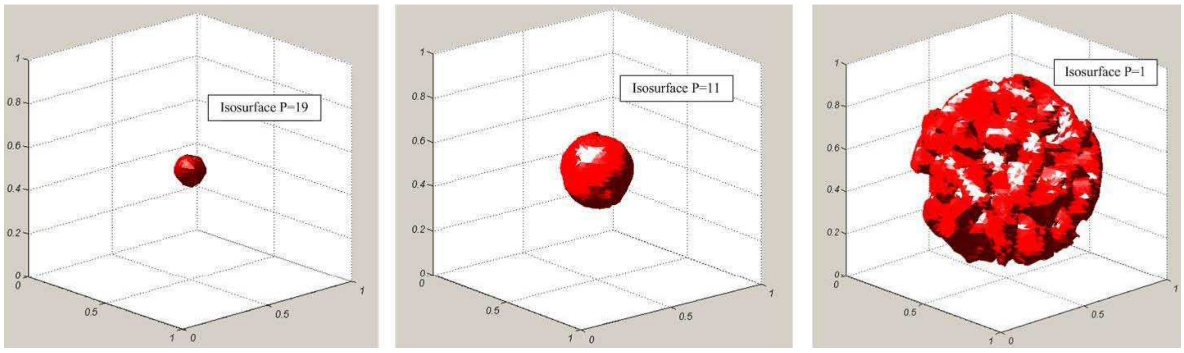


Figure 5.3: Simulation of gas expansion using kinetic Voronoi diagram, see [7].

5.4 Motion Interpolation

The problem of motion interpolation is slightly different than the other presented applications of kinetic data structures as it uses the kinetic approach for handling purely dynamic problem (speaking in terms of triangulation updating techniques as shown in Chapter 4). Let us have a sampled process that is represented by different states of Delaunay triangulation for individual samples at $t = \{t_0, t_1, \dots, t_n\}$ and it does not matter whether $t_{i+1} - t_i$ is constant or variable. In order to perform any sort of computation over the dataset at any given t_i, we need to change the triangulation from the state at the time t_{i-1} to the state at $t = t_i$. As the sampling frequencies are usually quite high in practise, we may expect that the states of the triangulation at t_i will be very similar to the state at t_{i+1} and thus it would be very convenient if this resemblance could be exploited in any fashion.

Guibas and Russel showed in [25, 46] that this can be done by using the idea that we are only provided with the states of the dataset at precisely given time instants and do not know anything about the data between these times. The lack of our knowledge inbetween the given times may even be exploited because the intermediate states will not be used for any kind of computation and we may thus manipulate them in any viable fashion. The most straighforward method of transforming the dataset given at t_i to the dataset given at t_{i+1} would be to remove the points from the old triangulation and reinsert them back at the coordinates that correspond with the new state. To do so, we might either discard the whole old triangulation and recreate it from the scratch or remove and reinsert the points one-by-one. However, the referred works show that both of these ideas are slower than using the kinetic approach.

There are two different ways of moving the points from their old positions to the new ones with greater efficiency than rebuilding the whole triangulation. We may either try to minimize the trajectory travelled by each of the points or try to minimize the polynomial degree in the certificate failure computation. If we choose to minimize the trajectories of the points, we move them along linear trajectories, we may even use the regular triangulation (see Chapter 2 and move the points along linear trajectories in the lifted space. The other approach, where the degrees of polynomials of the certificate functions are minimized consists of moving the points linearly along one axis at a time.

Russel shows in his PhD. thesis [46] that these methods themselves are not faster than the static updates due to the overhead of certificate failures computation, but they may be modified using so-called *filtering*. Filtering is based on the fact that we may use certain knowledge about the certificate functions to determine whether any failures worth computing will occur or not; this is done by using methods such as the Descartes' rule of sign, Sturm sequences and others (as described in Chapter 4. The kinetic based update techniques then outperform the static based methods.

5.5 Summary

The kinetic data structures (and kinetic Delaunay triangulations in particular) are a tool with a wide variety of possible uses – we described three of their practical applications covering three completely different areas and one general-use method. Clearly, the kinetic Delaunay triangulation may be used in any similar area such as air traffic, simulation of animal groups of various kinds and might even be exploited for wide range of mathematical problems that may be described with systems of partial differential equations and solved by finite element method. The comparison of triangulation updates using the dynamic and kinetic approach not only shows the potential of the kinetic approach to be faster in the field of sets of moving data but also demonstrates its extraordinal versatility.

Chapter 6

Contributions

This chapter will describe our method for handling the kinetic Delaunay triangulation. First, we show an overview of our method with a general description of the algorithm we use for computing the certificate failure times. After that, some general purpose optimizations in kinetic Delaunay triangulation handling are given, which enhance the theoretical understanding of the problematics and may be practically used to improve the stability and performance of various KDT implementations. Finally we will describe some of the applications of our implementation of the kinetic Delaunay triangulation.

6.1 Our Method for Kinetic Delaunay Triangulation Handling

Independentely on the research performed by Guibas and Karavelas in [27], we developed our own method that may be characterized as hybrid according to the terminology established in this paper (this classification only considers the method of the event times computation as we have already described in Chapter 4). It is important to note that KDT management includes several parts other than the event computation which determine its overall functionality and performance. This chapter describes our implementation of the problem of kinetic Delaunay triangulation management (note that our method has already been published in [51, 52, 54]).

Construction Methods and Data Structures

From the variety of methods available for Delaunay triangulation construction, we decided to use the incremental insertion algorithm with a walk-based triangle search. This algorithm has several very important features with respect to the kinetization principle. At first, it is very simple to implement, it is online (we may simulate discrete time movement by removing and reinserting the points in the triangulation) and the search algorithm allows us to change the triangulation structure with relative ease. This last feature is especially important, because the movement of the points will almost surely cause a large number of alternations in the triangulation. If we used a DAG-based or similar lookup structure, we would have to perform relatively large-scale alternations in its structure, which would be prone to errors. From the variety of possible walking algorithms, we have chosen to use an orthogonal walk algorithm in combination with the Remembering stochastic walk (see [35, 48]). The algorithms of these walks may be found in the aforementioned resources.

Point Movement

Our current implementation of KDT only allows point movement along linear trajectories without any acceleration, i.e., the time-dependent point coordinates are in the following form:

$$p_i(t) = (x_i(t), y_i(t))$$
$$x_i(t) = x_i(0) + \Delta x_i \cdot t; \; y_i(t) = y_i(0) + \Delta y_i \cdot t$$

Where $p_i(0) = (x_i(0), y_i(0))$ is the initial position of p_i and $\Delta x_i, \Delta y_i$ is the velocity of p_i in the x and y axis respectively. Our experiments, as well as some of other applications, suggest that this type of movement, however simple, is usable for a wide variety of problems – e.g., collision detection applications may use a piecewise linear movement approximation and the motion interpolation problem (as mentioned in Chapter 5) also uses various kinds of linear movement functions. We will show our own applications of KDT that use linear movement of the generating points later in this chapter.

Event Computation

As we have already shown in the previous text, the certificate function computed for the points p_i, p_j, p_k, p_l is in the form shown in Eq. 6.1 and since the linear point movement results in polynomial certificate functions of the fourth or lower degree, $c(t)$ is a polynomial function:

$$c(t) = \begin{vmatrix} x_i(t) & y_i(t) & x_i^2(t) + y_i^2(t) & 1 \\ x_j(t) & y_j(t) & x_j^2(t) + y_j^2(t) & 1 \\ x_k(t) & y_k(t) & x_k^2(t) + y_k^2(t) & 1 \\ x_l(t) & y_l(t) & x_l^2(t) + y_l^2(t) & 1 \end{vmatrix} \tag{6.1}$$

In order to compute the events, we have developed a special method for polynomial root finding. This method is general and technically able to solve polynomials of any given degree, but we have not tested it for this purpose yet as the applications of our method did not require this extension of functionality (since we only work with linear movement). It is based on the information that can be obtained about a polynomial from its Sturm sequence about the root locations and multiplicities. The roots themselves are then found by combining this information with numerical methods.

Sturm$_3$ algorithm shown in Alg. 6.1 describes the approach we use for solving polynomials of the third degree (finding the roots of higher order polynomials is very similar). The algorithm works in the following fashion: the Sturm sequence of the given polynomial $c(t)$ (i.e., the certificate function) is constructed and used to obtain two important pieces of information – the total amount of roots and all the multiple roots of this polynomial (as we already know from Chapter 4, both the exact locations and the multiplicities of the multiple roots may be obtained in this fashion). If any multiple roots exist, we divide $c(t)$ by the polynomial $(t - t_1)^{r_1} \cdot (t - t_2)^{r_2} \cdot \ldots \cdot (t - t_n)^{r_n}$ where $t_1, t_2, \ldots, t_n$ are the multiple roots with mulitplicities $r_1, r_2, \ldots, r_n$. If $c(t)$ does not have any multiple roots, it is left unmodified. Together with the fundamental theorem of algebra (see e.g., [44]), we are now able to solve $c(t)$ by using either the analytical formulas if the degree of the resulting polynomial is equal to one or two, or suitable numerical methods if the degree of $c(t)$ is greater. For the numeric part of the approach, we use the method of bisection for the initial root position estimation and Newton's method to enhance the precision of this estimation to the required value. We have chosen these methods because of their simplicity in the case of bisection and because of their extremely easy implementation and excellent expected performance in the case of Newton's method. We have also tried solving the lower degree polynomials numerically, but the results were no more precise than those obtained by the analytical approach and the analytical approach is very easy to use and fast since the analytical solutions are well known for the linear and quadratic functions (which are polynomials of the first and second degree).

Input:

- $c(t) = \sum_{i=0}^{3} a_i \cdot t^i = 0$ - a polynomial of the third degree.

Output:

- A set $\{t_i\}_{i=1}^{r}$ of real roots of $c(t) = 0$, $r \leq 3$.
- Or an empty set, if no real roots exist.

Auxiliary:

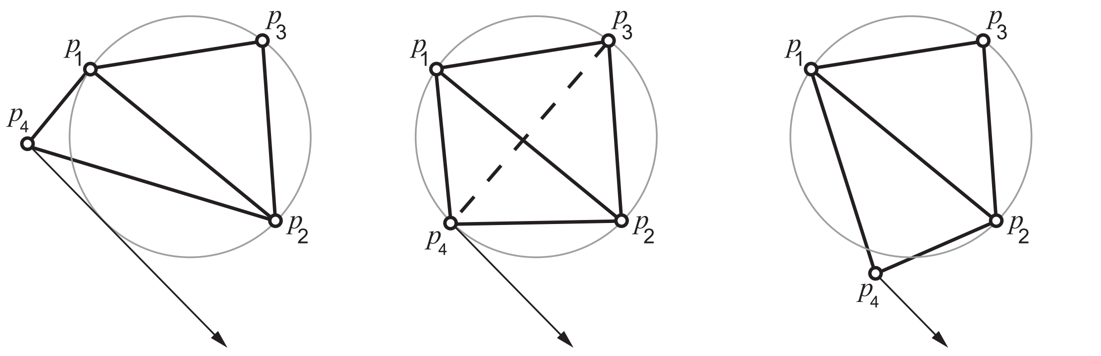
- Sturm sequence $f_1(t), ..., f_m(t)$ of the polynomial $c(t)$.
- $R_m = \{r_{mi}\}_{i=1}^{r_{mult}}$ - a set of all the multiple roots of $c(t)$.
  - Each multiple root r_{mi} is contained $m_i - 1$ times, where m_i is its multiplicity.
  - $R_m = \emptyset \Leftrightarrow c(t)$ has no multiple roots.

```
// Create the Sturm sequence

$f_1(t), ..., f_m(t) \leftarrow$ Sturm sequence of $c(t)$
$r \leftarrow (V(-\infty) - V(\infty))$// See Eqn. 4.7

if $r = 0$ then
    // $c(t)$ has no real roots

    // This situation cannot occur for the polynomials of the third
       degree, but is possible for the polynomials of even degree.
    Return empty set $\emptyset$ of roots
end
// Obtain the multiple roots of $c(t)$
$R_m \leftarrow$ set of $r_{mult}$ roots of $f_m(t)$
if $||R_m|| = 2$ then
    Return $\{r_{m1}, r_{m1}, r_{m1}\}$// One triple root

else if $||R_m|| = 1$ then
    // $c(t)$ has a double and a single root

    $r_s \leftarrow$ the only single root of $\frac{c(t)}{(t-r_{m1})^2} = 0$
    Return $\{r_{m1}, r_{m1}, r_s\}$// A double and a single root

else
    // No multiple roots, solve $c(t)$, using a suitable numerical method

    Return $\{r_1, r_2, r_3\}$ // Set of three distinctive roots.

end
```

Algorithm 6.1: Sturm$_3$ Algorithm.

6.2 Root Classification Research

As a part of our work, we have performed a research on the usability of the computed polynomial roots. The results of our research suggest that not all the roots of the certificate functions are usable for determining the times of the topological events. Let us consider the situation depicted in Fig. 6.1. In this figure, we may see that the point p_4 moves tangentially to the circumcircle of the triangle $p_1p_2p_3$.



(a) Initial situation. (b) Singular state. (c) Point p_4 moves forward without any changes in the topology.

Figure 6.1: Tangential movement of a point against the circumcircle of a triangle.

As we may see (if the points p_1, p_2, p_3 are static) there is only one certificate failure which is displayed as a singular state in Fig. 6.1(b). If we scheduled an event for this certificate failure, the triangulation will cease to be Delaunay and eventually it would even cease to be a triangulation as defined earlier because the edges would start to overlap. It is thus necessary to distinguish this type of events. The situation shown in Fig. 6.1 is the simplest possible case of the occurence of this type of event and it may distinguished by obtaining a double root of the certificate function. The following lemma shows how we can detect similar situations for more complicated point configurations as long as the movement of the points remain polynomial.

Lemma 6.2.1. *The roots of a polynomial certificate function $c(t)$ may be divided into two groups as follows:*

- *All roots of even multiplicity may be ignored when determining the times of topological events.*
- *All roots of odd multiplicity determine the time of a single topologic event.*

Proof. Let us rewrite the polynomial $c(t)$ as:

$$c(t) = (t - t_0)^r \cdot q(t) \tag{6.2}$$

where t_0 is a root of $c(t)$ with multiplicity r and $q(t)$ is a polynomial function. Let us now find an arbitrary small $\varepsilon > 0$ such that there will be an interval $I = (t_0 - \varepsilon; t_0 + \varepsilon)$ such that $q(t)$ has no roots in I. Also, let us define $c_0(t) = (t - t_0)^r$.

If t_0 is of even multiplicity, we may say that $r = 2k$ and thus:

$$c_0(t) = \left[(t - t_0)^2\right]^k$$

We may see that $\forall t \in \mathbb{R} : c_0(t) \geq 0$ and since $q(t)$ does not have any roots in I (and thus the sign of its value does not change over I because it is a continuous function), we may clearly see that the sign of $c(t)$ does not change over I (if the zero at $t = t_0$ is ignored). Moreover, we may see that the certificate with the certificate function $c(t)$ does not fail for any time $t \in I$ and since t_0 is the only root of $c(t)$ in I, it does not mark a certificate failure.

If t_0 is of odd multiplicity, r can be expressed as $r = 2k + 1$ and we have:

$$c_0(t) = \left[(t - t_0)^2\right]^k \cdot (t - t_0)$$

We may see that the sign of $c_0(t)$ does change exactly once in the interval I and thus the sign of $c(t)$ changes too and the certificate function fails, determining the time of a single topologic event. □

As a result of the presented knowledge, we may state that any method which allows us to easily determine the multiplicity of the polynomial roots is highly valuable. Also, the hybrid method presented by Guibas and Karavelas in [27] is not entirely correct. Prior to the actual root computation, their method replaces each certificate function $c(t)$ with other polynomial function $d(t)$ which has the same roots but all with multiplicities equal to one. As shown in the example in Fig. 6.1, this may lead to the execution of nonexistent topologic events, topology distortion and increase in numerical instability.

6.3 Redundant Event Reduction

As shown in Chapter 4, some of the topologic events that are computed will be descheduled before their execution and the runtime spent on their computation is wasted. We call these events redundant (as shown in the previous text – in Chapter 4). Our research shows that these events represent roughly 50% of all the events that are computed during the application runtime as can bee seen in the graph in Fig. 6.2.

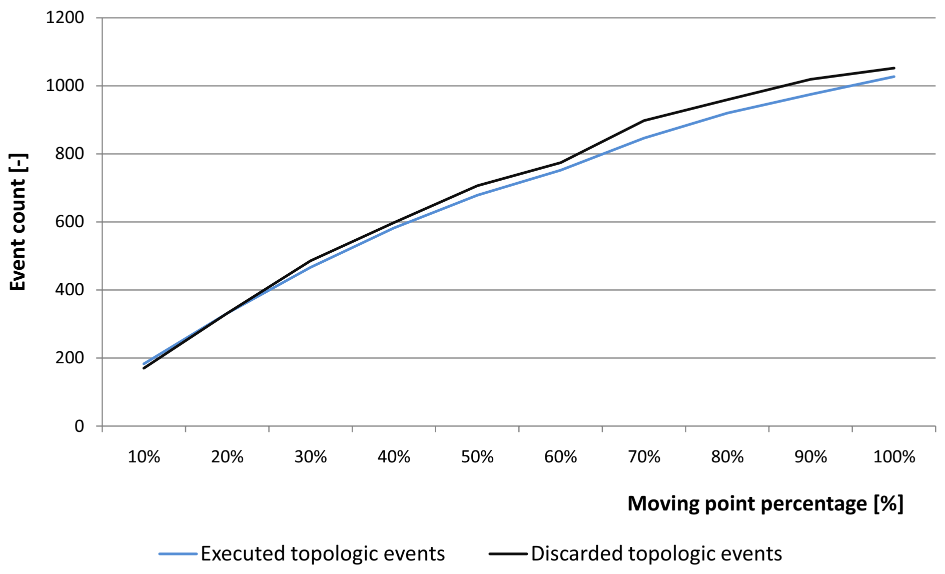


Figure 6.2: Total amount of scheduled and discarded topologic events.

The graph in Fig. 6.2 shows the total amount of scheduled topologic events for various percentages of moving points in the generator set. The test was concluded on a set of 100 points that were randomly placed in a square-shaped area and a certain subset of these points was assigned a random velocity vectors. After ten seconds of KDT management, we were able to determine that the total number of succesfully processed topologic events is nearly equal to the number of the discarded events which means that approximately half of the time spent on the event computation is lost. Furthermore, as we can see in the graph in Fig. 6.3, the runtime spent on the event computation represents over 90% of the total time needed for the KDT manegement if at least 40% of the points are moving.

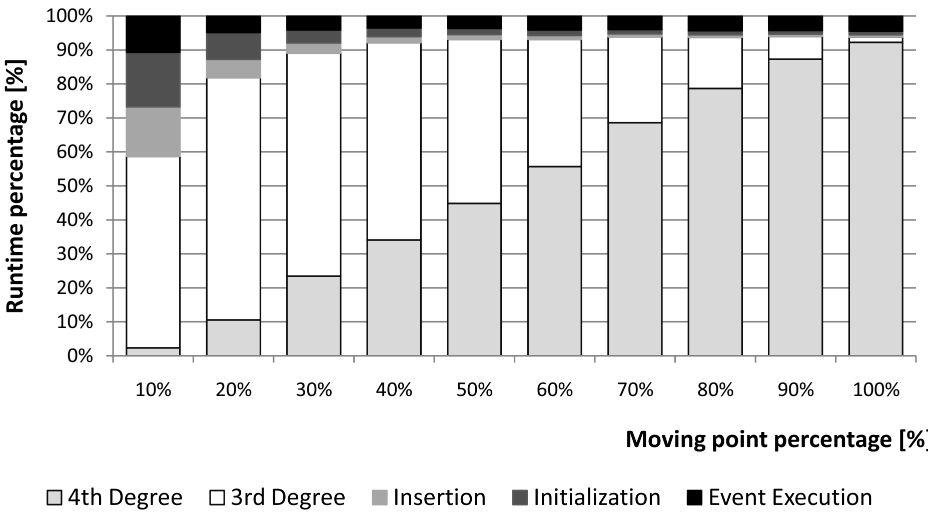
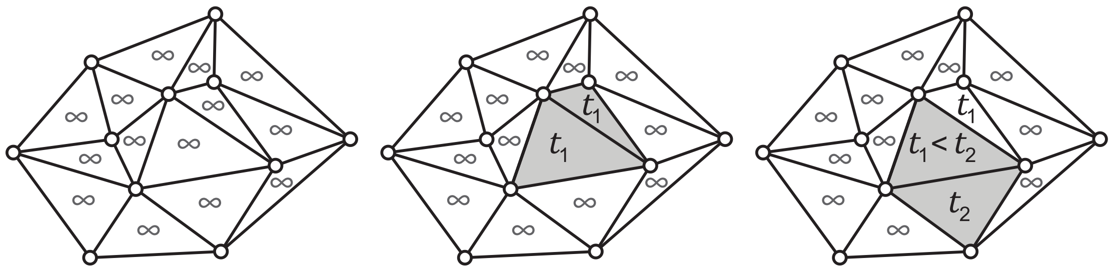


Figure 6.3: Distribution of runtime consumption during KDT management.

As shown in the previous text, topologic event computed for triangles $p_ip_jp_k$ and $p_ip_jp_l$ (the edge common to these two triangles being p_ip_j) at the time t_e will become redundant if at least one of these two triangles is removed from the triangulation at the time $t_r < t_e$ (such a removal will usually occur as a result of an edge flip). With a relativelly little effort, this knowledge may be used to reduce the number of computed events by assigning a time of the next event scheduled for that triangle to each triangle in KDT or an infinite value if no event is scheduled for the triangle.

Let us first consider the initialization step as shown in Alg. 4.1. At the beginning of this step, we have a Delaunay triangulation and no computed topologic events. We now assign an infinite value to each triangle as a time of the next topologic event that will alter this triangle – see Fig. 6.4(a). Following the aforementioned algorithm, we now choose pairs of adjacent triangles and try to compute the first topologic event that will occur for the two chosen triangles. If such an event exists and will occur at the time t_1, we push it into the priority queue and assign the value t_1 to these two triangles as shown in Fig. 6.4(b) (the triangles are marked by grey color in the figure). Later on, we may try to compute a topologic event for a triangle pair which contains a triangle with a non-infinite time value as shown in Fig. 6.4(c). Let us say such an event exists and will occur at the time t_2. We will then assign the value of $\min\{t_1, t_2\}$ to each of the two triangles in the currently handled triangle pair, where t_1 is the original time value assigned to the triangle. In this case, the redundant topologic event is not computed and it is thus necessary to combine this approach with such a computation method which allows us to compute the topologic events separately for given time intervals.

The situation during the iteration step is very similar – when we handle a topologic event, we need to compute new event times for the newly created triangles and their neighbours. We proceed in a very similar fashion, i.e., we are only interested in the events that will occur before the time given by the minimum value assigned to the currently considered triangle pairs (in this case, the triangles newly created by the edge flipping are assigned an infinite value).



(a) Start of the initialization step. (b) First event computed. (c) Redundant event detected.

Figure 6.4: Initialization step with redundant event reduction.

The graph shown in Fig. 6.5 demonstrates the changes in the number of computed redundant topologic events if we use the aforementioned method. As we can see, the number of redundant topologic events drops approximately to one half of the original number. We may see that only one third of the computed topologic events is now being discarded as redundant events (in other words, we are able to detect approximately 30% of the redundant events before they are computed).

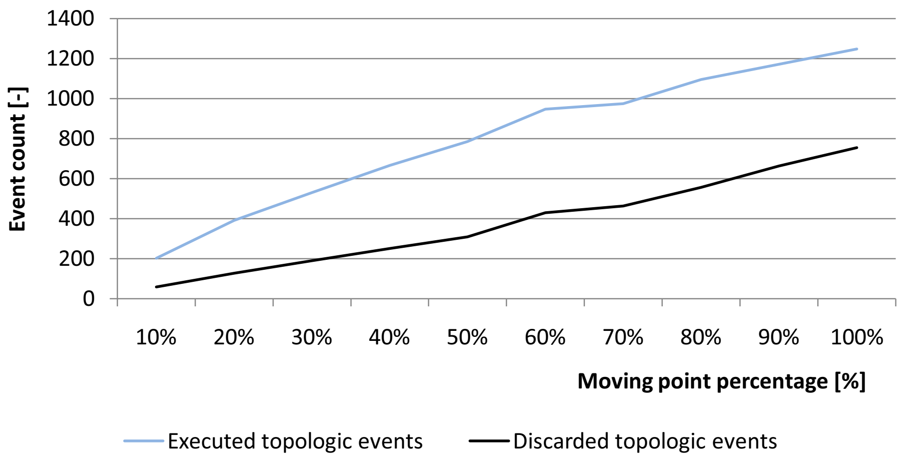


Figure 6.5: Total amount of scheduled and discarded topologic events when using redundant event reduction.

As this process leads to the reduction of the amount of the most time consuming tasks in the management of KDT, we may expect a significant reduction in the runtime consumption. On the other hand, the drawback of this approach is a slightly increased numerical instability which may lead to missing some of the non-redundant topologic events.

6.4 Applications of Kinetic Delaunay Triangulations

6.4.1 Video Representation

As a part of our research, together with *Petr Puncman*, we have developed a method for video representation based on kinetic Delaunay triangulations (presented in [55]). Unfortunately, our research in this area has been suspended because our co-researcher Petr Puncman, whose research concentrated on the video representation part of the problem, left our research team. Illustrational example of our approach is shown in Fig. 6.6 where we can see three different stages of our method. In the left part, the kinetic Delaunay triangulation constructed for a chosen subset (see further) of pixels in the image is shown. In the middle part, the velocity vectors describing the movement of the points are shown (these vectors are computed by matching the positions of the given points in two different frames of the video). The right part of Fig. 6.6 shows the moving part of the KDT after the points have been moved.

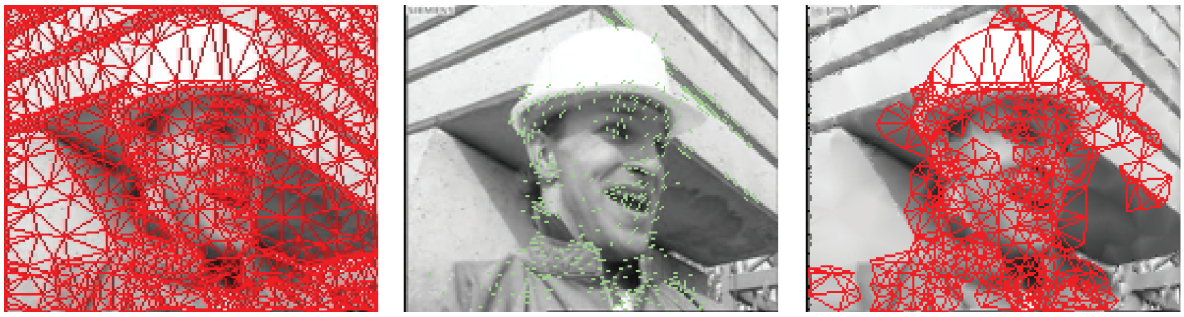


Figure 6.6: Illustrational example of our approach to video representation using KDT, [55].

In our approach, we focus on a combination of various methods for video manipulation together with the abilities of KDT. That means that we perform both sampling of the pixels of the original image in coder and interpolation or warping of the full scene by using KDT. The idea of video representation by KDT is based on a creation of KDT and representing the changes in the video by the movement of the KDT vertices. In the first step, the relevant points and some random points have to be obtained from a key frame[1]. These points then define the KDT until a new picture group is formed, starting with the next key frame. The second step means processing frames between the two following key frames to form interlying frames by moving the points in the triangulation and reconstructing the frames from the current state of the triangulation. The movement of the points in KDT is defined by the vectors obtained during motion estimation. So the whole process may be described as follows:

Important point selection: a set of important points is selected from the input image and KDT is created from these points. To obtain valuable samples from the input frame, we mix edge points with random points at the ratio of 1:1 (the best ratio value was obtained from the performed ratio tests). Starting with an already given point count, we select the most suitable points from the frame by the methods of discrete convolution, selective thresholding and randomized selection. The resulting set of pixels then in the ideal case contains a predefined amount of points.

[1]Key frames are certain important frames which are commonly used in almost any compressed video sequence, these frames are compressed basically in the same way as the static images. Also, the frames lying between two keyframes are compressed using the information in the key frames.

These points should be distributed densely on the edges, sparsely in the homogenous areas and uniformly everywhere else in the image.

Motion estimation: corresponding blocks centered around each vertex are found in consecutive frames. Their positions are then used to define motion vectors. Although the block matching algorithm often deals with macroblocks covering the whole frame, in our case we had to find which block in the next frame makes the best match for the block centered around each KDT point in the current frame. For the evaluation of differences between the blocks we have used various different metrics. Once all the vectors in a frame are known, velocities of their points in KDT are set.

KDT movement: the movement is initialized transforming the current frame into the following one using the methods described in the previous text. One thing worth noting is the fact, that the points obtained in the *important point selection* step lie on a square grid (they are image pixels) and thus tend to be found in various singular configurations (multiple cocircular points etc.). We introduce small random variations in the velocity vectors of these points which help to overcome these singularities without any loss of precision (the points finish their movement in the correct pixel positions after rounding).

Video decoding: a frame is reconstructed from the data stored in the KDT. The previous steps provided us with the most important points in each frame and their inter-frame correspondency thus allowing us to compute the motion vectors for these points. By inserting the points obtained in each key frame into a KDT and moving them along the motion vectors we get a sequence of triangulation states that represent each frame between two keyframes. From these triangulations we must now decode the approximation of the original frame. The use of DT is crucial for our method if we want to obtain usable compression ratio. It is vital to note that for every other type of triangulation we have to store not only the coordinates and color of each vertex but also an information on the surrounding topology. On the other hand in DT we may be sure that if no four points in the triangulation are cocircular, we will always reconstruct the same triangulation (independently on the algorithm used) without storing any additional topology data. As it has been shown (see [43]) the compression may be expected to be meaningful (considering a method without any stored topology information) if we construct the KDT using generator sets of a size up to 20% of the number of pixels in each video frame if we use a trivial compression method and up to 30% if a more sophisticated method is used.

6.4.2 Early Warning System for Air Traffic Control

Recently, we have begun to develop a system for early warning for the use by the air traffic control. The problem basically represents a collision detection with the feature that the upcoming collisions have to be detected with sufficient advance so the aircrafts may take some kind of evasive maneuver to leave the collision course. The application is similar to the one presented in [24] for marine environment but poses several key differences.

Currently, the air traffic is monitored and controlled mostly by human operators. Each of these operators obtains a certain (rather small) subset of the air traffic data provided by a tracking device such as radar, etc. Furthermore, the aircrafts may usually only move through strictly defined corridors called routes, which are defined differently for different kinds of aircrafts (e.g., helicopter routes may differ from international flight routes). The routes also differ in accessibility – certain routes are only available for some heights, specific visibility conditions, etc. The predefined heights on which the aircrafts travel along the routes are called flight levels, which vary with changing atmospheric pressure. This effect is caused by the definition of the flight levels. The flight level is a level of constant atmospheric pressure which causes the aircrafts to flight in different heights for different atmospheric conditions even though the aircrafts are located on the same flight level. Further information on the routes, flight levels and other air traffic rules may be obtained in [2].

Our approach provides a global perspective of the problem. By analyzing the whole set of aircrafts detected by the radar at once, we are able to detect potential threats before any other postprocessing is applied to this set and to detect such situations when two aircrafts are on a collision course. The overall functionality of our approach may be divided as shown in Fig. 6.7. We may see that our methods consists of the following parts: obtaining the air traffic data from a radar (1), mapping the data to a plane (2), construction of the KDT (3), and using the KDT for collision detection (4).

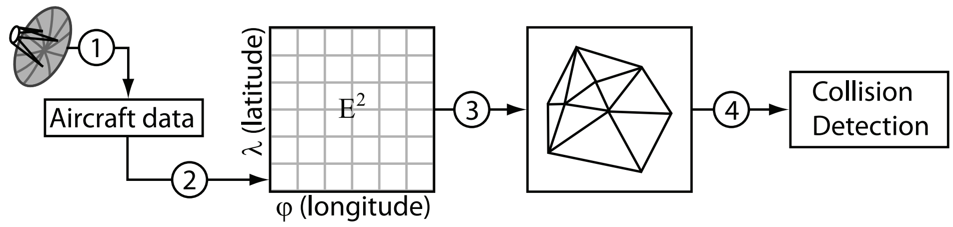


Figure 6.7: Block scheme of early warning system for air traffic control.

Data reading: the data set we used in our application was obtained from the Air Navigation Services of the Czech Republic. It was generated from a record obtained by logging real radar entries during standard traffic. Upon detection, the radar data was processed by a tracker device which supplied the individual records (among others) with aircraft-unique identifiers thus ensuring that when a single aircraft appears more than once in the record, it will be uniquely identifiable each time it reappears.

Mapping to E^2: for the purposes of our application, we used a simple planar projection using *WGS84* model of the Earth for the coordinate transformation (see [39]). *WGS84* is the most commonly used model in the field of air traffic control. Another problem related with the aircraft mapping to 2D is that the aircrafts do not fly along linear trajectories (especially when waiting above an airport, the aircraft may be flying in circles for a prolonged period of time). Using the newest available data from the radar readings for each aircraft and one of the mentioned Earth models, we predict the next position of the aircrafts by using a simple linear extrapolation.

KDT management and collision detection: the KDT management in this particular application differs a little from the other presented applications, because, aside from the topological events, the toplogy of the triangulation may change when a new information about an aircraft is obtained from the radar due to the extrapolation of the positions of the aircrafts. At this moment, we may update our triangulation by removing the given point from it, reinserting it back on its correct position and recomputing the topological events in the changed area or allow some relative tolerance τ for the position vector of the aircrafts and only update its velocity vector and recompute the topological events on the neighbourhood – the aircraft positions are then repaired by the remove-reinsert method only when they are too far away from the positions detected by the radar (see Fig. 6.8).

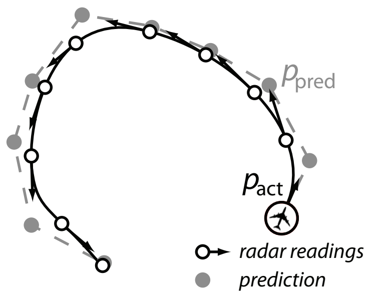


Figure 6.8: The difference between the real path of an aircraft (black line, empty circles) and the extrapolated path with some position tolerance (gray dashed line, filled circles).

Experiments with our application show that it is both sufficiently robust and fast to be used for the air traffic over the Czech Republic. Furthermore, we anticipate it to be usable even for such datasets which would include more than ten times the amount of the aircrafts present in the air space of the Czech Republic. On the other hand, if we consider the current approach in the field of air traffic management, our application represents completely different approach than the one which is practically used and thus it may be seen as a possible way of future development as it provides the global point of view to the problematics and a way of detecting the upcoming threats automatically.

Chapter 7

Future Work

Prior to the description of the further areas of advancement in our work, let us now briefly summarize its current state. Using the methods described in the previous text, we were able to implement a library for kinetic Delaunay triangulation management based on a hybrid computational method for the certificate failures computation. By employing the aforementioned theoretical improvements in the implementation, we are now able to use our library as a tool which is usable for real life problems and is both sufficiently fast and numerically stable for this purpose. As a part of our research, we have also identified three different possible areas of future progression which include the video representation by KDT (as described in Chapter 6), the early warning system in the field of the air traffic control (also described in Chapter 6) and the crowd simulations for virtual reality (VR) cities (the usability of KDT for this problem is shown in Chapter 5). The possibilities of progress in these areas will be evaluated in the following text.

Image Representation using KDT

In our previous work together with Petr Puncman, we have shown that the area of video compression methods represents a viable application of KDT as the Delaunay triangulation may be efficiently used as the main means of the compression of a static image since it is not necessary to store any additional information about its topology (DT is unique for any given set of points). Also, since the changes in the video may be seen as a movement of the individual pixels, we may exploit the KDT by applying this movement to the generator set.

Our research in this area is currently suspended as we are looking for a student who would continue in the work of Petr Puncman because the amount of work which needs to be done in the video representation and compression part of the problematics is comparable to the amount of work in the field of the kinetization of this problem. Furthermore, the video problem of video representation itself is entirely out of scope of the kinetic data structures. On the other hand, it represents an interesting challenge and should provide a good basis for a master thesis in this area.

Early Warning System in Air Trafic Control

Our experiments in this field have shown a general usability of KDT as a base data structure for the early warning system in the air traffic control, however, there are several optimization tasks and domain-specific features left to implement such as flight level and flight routes handling, etc. As the general usability of the kinetic data structures in similar applications has been already shown by other researches, the only problems left to solve in this branch of our research are almost purely of implementation and testing nature. Considering these facts, we have evaluated the remaining work to be most suitable for an undergraduate student and could ideally serve as a focus of a bachelor thesis.

Crowd simulation for VR cities

Unlike the previous two research area, the problem of crowd simulations has not been discussed in Chapter 6 because we are currently gathering the necessary knowledge and designing the optimal use of the kinetic Delaunay triangulation. Our analysis of the problem (see [53]) shows that the main problem in the field of crowd simulation for the purposes of virtual reality is that (if the underlying data structure is graph-based) it most commonly uses two different structures. One data structure for the global navigation of the pedestrians and the other for the local collision avoidance (see the general description in Chapter 5). Note that this analysis is only available online as we consider its text to be too extensive and slightly out-of-scope to be included in this work.

The main direction of our research in this area will be the development of a kinetic data structure (based on KDT) which would be usable for both the afforementioned tasks (i.e., the global navigation of pedestrians and local collision avoidance). Such a data structure should be usable by various path-planning algorithms while maintaining the relations between the individual pedestrians in the kinetic fashion in order to employ some short-range collision avoidance methods. For the purposes of development of this data structure, we plan to use *VRECKO: Virtual Reality Framework* (which is being developed in the Human-Computer Interaction Laboratory at the Masaryk University in Brno) as the testing environment.

Chapter 8

Conclusion

This work presents a compact overview of the kinetic Delaunay triangulations and similar data structures (represented by kinetic Voronoi diagrams and regular triangulations, which may be used in a very similar fashion). Several possible ways of constructing a Delaunay triangulation were discussed with the focus on the usability for further kinetization of the constructed data structure. Regular triangulations were considered as well, as some applications may benefit strongly by using the weighted points and the principle of kinetization is very similar.

Further, the general approaches for handling the kinetic data are shown, together with the most commonly used properties for evaluation of these structures. The difference between kinetic and dynamic approach to movement is described with the key features of each of these methods. Also the concept of certificate functions and certificate failures (i.e., the kinetic events) is described in general and with special respect paid to the kinetic Delaunay triangulation, where a concrete method for handling this particular data structure is described. Also the most important problem in the field of kinetic data structures is discussed in detail – the computation of the kinetic events. General categorization of the available methods is shown, together with the most commonly used methods for polynomial equation solving (as the certificate functions are most commonly polynomial functions).

Several representants of the applications that use kinetic Delaunay (or regular) triangulations are also shown – we may see a wide variety of applications that benefit from using this data structure ranging from collision detection through various types early warning systems for different environments, navigation of pedestrians in artificial crowds, to mathematical simulations of fluid dynamics. The kinetic data structures have also been extensively compared with the kinetic data structures and this work presents the results of these comparisons.

The next part of this work presents the author's contributions which consist of an implementation of planar kinetic Delaunay triangulation. Our implementation is currently being used as a geometic means of image representation in the application developed in cooperation with *Petr Puncman* (a former member of the University of West Bohemia) and in the application for early warning in the air traffic which is being developed with assistance of the *CS Soft Group* company. We also presented some theoretical enhancements of the kinetic Delaunay triangulation which may be used to improve the overall performance of various implementations of this data structures.

Future progress of the author's work is also presented. Three different fields of study are described and analyzed. Two of these fields (the video represantion and the early warning system in the air traffic control) are shown to be potentially usable as foundations of students' work while the third field (the simulation of virtual crowds) gives the outline of the author's future research.

Acknowledgements

We would like to thank to Mr. Karel Hanton and Mr. Petr Mach from the *CS Soft group* who provided us with the air traffic data, expert consultations on the topic of air traffic control, information about the practise in this field and valuable references to various information sources.

Bibliography

[1] ABRAMOWITZ, M., AND STEGUN, I. A. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, ninth dover printing, tenth gpo printing ed. Dover, New York, 1964.

[2] AIR NAVIGATION SERVICES OF THE CZECH REPUBLIC, AERONAUTICAL INFORMATION SERVICE. Various documents. Available online - http://lis.rlp.cz.

[3] ALBERS, G., GUIBAS, L. J., MITCHELL, J. S. B., AND ROOS, T. Voronoi diagrams of moving points. *International Journal of Computational Geometry and Applications 8*, 3 (1998), 365–380.

[4] BARBER, C. B., DOBKIN, D. P., AND HUHDANPAA, H. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw. 22*, 4 (1996), 469–483.

[5] BASCH, J., GUIBAS, L. J., AND HERSHBERGER, J. Data structures for mobile data. *Journal of Algorithms 31*, 1 (1999), 1–28.

[6] BASCH, J., GUIBAS, L. J., SILVERSTEIN, C. D., AND ZHANG, L. A practical evaluation of kinetic data structures. In *In Proc. 13th Annu. ACM Sympos. Comput. Geom* (1997), ACM Press, pp. 388–390.

[7] BENI, L. H. *Development of a 3D Kinetic Data Structure Adapted for a 3D Spatial Dynamic Field Simulation*. PhD thesis, Université Laval, Québec, 2009.

[8] CARETTE, V., MOSTAFAVI, M., AND DEVILLERS, R. Towards marine geographic information systems: Multidimensional representation of fish aggregations and their spatiotemporal evolutions. pp. 1 –10.

[9] CIGNONI, P., MONTANI, C., AND SCOPIGNO, R. Dewall: A fast divide and conquer Delaunay triangulation algorithm in E^d. *Computer-Aided Design 30*, 5 (1998), 333–341.

[10] DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. *Computational geometry, algorithms and applications*. Berlin Heidelberg: Springer, 1997.

[11] DELAUNAY, B. N. Sur la sphère vide. *Bulletin of Academy of Sciences of the USSR*, 6 (1934), 793–800.

[12] DEVILLERS, O. Improved incremental randomized Delaunay triangulation. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry* (New York, NY, USA, 1998), ACM, pp. 106–115.

[13] Devillers, O. On deletion in Delaunay triangulations. In *Symposium on Computational Geometry* (1999), pp. 181–188.

[14] Devillers, O., and Devillers, O. Improved incremental randomized Delaunay triangulation, 1997.

[15] Devillers, O., Pion, S., and Teillaud, M. Walking in a triangulation. In *SCG '01: Proceedings of the seventeenth annual symposium on Computational geometry* (New York, NY, USA, 2001), ACM, pp. 106–114.

[16] Edelsbrunner, H., and Seidel, R. Voronoi diagrams and arrangements. In *SCG '85: Proceedings of the first annual symposium on Computational geometry* (New York, NY, USA, 1985), ACM, pp. 251–262.

[17] Edelsbrunner, H., and Shah, N. R. Incremental topological flipping works for regular triangulations. *Algorithmica 15*, 3 (1996), 223–241.

[18] Ellenberger, K. W. Algorithm 30: numerical solution of the polynomial equation. *Commun. ACM 3*, 12 (1960), 643.

[19] Erickson, J., Guibas, L. J., Stolfi, J., and Zhang, L. Separation-sensitive collision detection for convex objects. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 1999), Society for Industrial and Applied Mathematics, pp. 327–336.

[20] Ferrez, J.-A. *Dynamic Triangulations for Efficient 3D Simulation of Granular Materials.* PhD thesis, cole Polytechnique Fdrale De Lausanne, 2001.

[21] Fortune, S. A sweepline algorithm for Voronoi diagrams. *Algorithmica 2* (1987), 153–174.

[22] Gavrilova, M., Rokne, J., and Gavrilov, D. Dynamic collision detection in computational geometry. In *12th European Workshop on Computational Geometry* (Munster, Germany, 1996), pp. 103–106.

[23] Goldenstein, S., Karavelas, M. I., Metaxas, D. N., Guibas, L. J., Aaron, E., and Goswami, A. Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers & Graphics 25*, 6 (2001), 983–998.

[24] Goralski, I. R., and Gold, C. M. Maintaining the spatial relationships of marine vessels using the kinetic Voronoi diagram. In *ISVD '07: Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 84–90.

[25] Guibas, L., and Russel, D. An empirical comparison of techniques for updating Delaunay triangulations. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry* (New York, NY, USA, 2004), ACM, pp. 170–179.

[26] Guibas, L. J. Kinetic data structures: a state of the art report. In *WAFR '98: Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective* (Natick, MA, USA, 1998), A. K. Peters, Ltd., pp. 191–209.

[27] Guibas, L. J., and Karavelas, M. I. Interval methods for kinetic simulations. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry* (New York, NY, USA, 1999), ACM, pp. 255–264.

[28] Guibas, L. J., Xie, F., and Zhang, L. Kinetic collision detection: Algorithms and experiments. In *ICRA* (2001), pp. 2903–2910.

[29] Heigeas, L., Luciani, A., Thollot, J., and Castagné, N. A physically-based particle model of emergent crowd behaviors, 2003.

[30] Hirst, H. P., and Macey, W. T. Bounding the roots of polynomials. *The College Mathematics Journal 28*, 4 (1997), 292–295.

[31] Hjelle, O., and Dæhlen, M. *Triangulations and Applications*. Berlin Heidelberg: Springer, 2006.

[32] Joe, B. Construction of three-dimensional Delaunay triangulations using local transformations. *Comput. Aided Geom. Des. 8*, 2 (1991), 123–142.

[33] Kamphuis, A., and Overmars, M. H. Finding paths for coherent groups using clearance. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2004), Eurographics Association, pp. 19–28.

[34] Kohout, J. *Parallel Delaunay triangulation in 2D and 3D, Diplomová práce*. University of West Bohemia, Pilsen, Czech Republic, 2002.

[35] Kolingerová, I. A small improvement in the walking algorithm for point location in a triangulation. In *22nd European Workshop on Computational Geometry* (March 2006), pp. 221–224.

[36] Ledoux, H., and Gold, C. M. Modelling three-dimensional geoscientific fields with the Voronoi diagram and its dual. *Int. J. Geogr. Inf. Sci. 22*, 5 (2008), 547–574.

[37] Mostafavi, M. A., Gold, C., and Dakowicz, M. Delete and insert operations in Voronoi/Delaunay methods and applications. *Comput. Geosci. 29*, 4 (2003), 523–530.

[38] Mücke, E. P., Saias, I., and Zhu, B. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *SCG '96: Proceedings of the twelfth annual symposium on Computational geometry* (New York, NY, USA, 1996), ACM, pp. 274–283.

[39] National Imagery and Mapping Agency, DoD. World geodetic system 1984, its definition and relationship with local geodetic systems, technical report 8350.2. Tech. rep., 1997. http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf.

[40] Okabe, A., Boots, B., Sugihara, K., and Chiu, S. N. *Spatial tessellations: Concepts and applications of Voronoi diagrams*, 2nd ed. Probability and Statistics. Wiley, NYC, 2000.

[41] Overmars, M. Practical algorithms for path planning and crowd simulation. In *Proceedings of the 25th European Workshop on Computational Geometry (EuroCG)* (Brussels, Belgium, March 2009), p. 263. Invited speech.

[42] Press, W., Flannery, B., Teukolsky, S., and Vetterling, W. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[43] PUNCMAN, P. *Použití triangulací pro reprezentaci videa, Diplomová práce*. University of West Bohemia, Pilsen, Czech Republic, 2008.

[44] RALSTON, A. *A First Course in Numerical Analysis*. McGraw-Hill, Inc.: New York, 1965.

[45] ROUILLIER, F., AND ZIMMERMANN, P. Efficient isolation of polynomial's real roots. *J. Comput. Appl. Math. 162*, 1 (2004), 33–50.

[46] RUSSEL, D. *Kinetic data structures in practice*. PhD thesis, Stanford, CA, USA, 2007. Adviser-Guibas, Leonidas.

[47] SHEWCHUK, J. R. Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry* (New York, NY, USA, 2000), ACM, pp. 350–359.

[48] SOUKAL, R. *Aplikace algoritmu procházky v počítačové grafice, Diplomová práce*. University of West Bohemia, Pilsen, Czech Republic, 2008.

[49] SUD, A., ANDERSEN, E., CURTIS, S., LIN, M., AND MANOCHA, D. Real-time path planning for virtual agents in dynamic environments. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes* (New York, NY, USA, 2008), ACM, pp. 1–9.

[50] TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. Continuum crowds. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), ACM, pp. 1160–1168.

[51] VOMÁČKA, T. Delaunay triangulation of moving points. In *Proceedings of the 12th Central European Seminar on Computer Graphics* (2008), pp. 67–74.

[52] VOMÁČKA, T. Delaunay triangulation of moving points in a plane. Master's thesis, University of West Bohemia, Univerzitní 22, Pilsen, Czech Republic, 2008.

[53] VOMÁČKA, T. Terrain representation for artificial human navigation. Tech. rep., 2010. http://graphics.zcu.cz/Download-document/136-76_terrainRepresentation.

[54] VOMÁČKA, T., AND KOLINGEROVÁ, I. Computation of topologic events in kinetic Delaunay triangulation using sturm sequences of polynomials. In *SIGRAD 2008* (Linköping, 2008), University Electronic Press, pp. 57–64.

[55] VOMÁČKA, T., AND PUNCMAN, P. A novel video compression scheme based on kinetic Delaunay triangulation. In *Algoritmy 2009 : 18th Conference on Scientific Computing* (Bratislava, 2009), Slovak University of Technology, pp. 372–381.

[56] ŽALIK, B., AND KOLINGEROVÁ, I. An incremental construction algorithm for Delaunay triangulation using the nearest-point paradigm. *Int.J. Geographical Information Science 17*, 2 (2003), 119–138.

Activities

Reviewed Publications

- Tomáš Vomáčka and Petr Puncman. A novel video compression scheme based on kinetic Delaunay triangulation. In *Algoritmy 2009: 18th Conference on Scientific Computing*, pages 372-381, Bratislava, 2009. Slovak University of Technology.
- Tomáš Vomáčka and Ivana Kolingerová. Computation of topologic events in kinetic Delaunay triangulation using sturm sequences of polynomials. In *SIGRAD 2008*, pages 57-64, Linköping, 2008. University Electronic Press.

Student Publications

- Tomáš Vomáčka. Delaunay triangulation of moving points in a plane. *Master thesis*, supervisor Ivana Kolingerová, University of West Bohemia, Univerzitní 22, Pilsen, Czech Republic, 2008.
- Tomáš Vomáčka and Ivana Kolingerová. Delaunay triangulation of moving points. *In Proceedings of the 12th Central European Seminar on Computer Graphics*, pages 67-74, 2008.
- Tomáš Vomáčka and Ivana Kolingerová. Využití prioritní fronty pro správu Delaunayovy triangulace nad kinetickými daty. *Studentská vědecká konference*, 2007. Pilsen, Czech Republic.

Related Talks

- *Terrain Representation for Artificial Human Navigation.* Center of Computer Graphics and Data Visualization, University of West Bohemia, Czech Republic, March 2010.
- *Triangulating the Kinetic Data.* Center of Computer Graphics and Data Visualization, University of West Bohemia, Czech Republic, April 2009.
- *Delaunay Triangulation of Moving Points.* University of Maribor, Slovenia, November 2008.

Stays Abroad

- University of Maribor, Slovenia. November 2008, 1 week.

Participations in Projects

- *Triangulated Models for Haptic and Virtual Reality*. The Grant Agency of the Czech Republic, Project No. GAČR 201/09/0097.
- *Alternative Representation of Image Information Using Triangulations, junior research project*. The Grant Agency of the Czech Republic, Project No. KJB101470701.
- *CPG - Center of Computer Graphics - National Network of Fundamental Research Centers*. The Ministry of Education, Youth and Sports of the Czech Republic, Project No. LC 06008.