University of West Bohemia
Faculty of Applied Science
Department of Computer Science and Engineering

Diploma Thesis

Delaunay Triangulation of Moving Points in a Plane

Pilsen, 2008

Tomáš Vomáčka

Abstract

Delaunay triangulation of moving points (also called the kinetic Delaunay triangulation) is a time-dependent data structure intended to retain the Delaunay property despite the movement of the underlying points. This kind of structure may be found useful in various areas of computational geometry and computer graphics, including for instance collision detection, path planning, terrain deformations and others.

For managing the triangulation, we use the method of continuous legalization of the triangulation by computing the times of topological events (previously described by Gavrilova, Roos, Rokne and others). In order to compute these times, polynomials of various degrees have to be computed. We propose our own hybrid numerical-analytical method based on Sturm sequences of polynomials.

We also propose some methods for enhancing the boundaries of the movement. We consider the ways of exploiting the "linear movement only" restriction in order to simulate nonlinear movement, bounding points to clusters that share a common velocity and finally we consider the possibilities of projecting the planar data to three dimensions.

Poděkování

Na tomto místě bych rád poděkoval především Doc. Dr. Ing. Ivaně Kolingerové za trpělivé vedení, poskytnuté teoretické zázemí a cenné rady, kterými mě vedla během procesu tvorby této diplomové práce. Neméně důležité poděkování náleží mým rodičům za podporu, které se mi od nich dostávalo po celou dobu mých dosavadních studií a zejména potom mému otci, jemuž bych chtěl poděkovat za řadu velmi cenných rad a připomínek, které v nezanedbatelné míře pomáhaly formovat matematickou stránku mé práce.

Declaration

I hereby declare that this diploma thesis is completely my own work and that I used only the cited sources.

Pilsen, ...

Tomáš Vomáčka, ...

Contents

1 Introduction

Since its invention in 1934, Delaunay triangulation (and its dual Voronoi diagram) has been used in various fields of computer graphics and computational geometry. Examples of its use include terrain modelling, building meshes for the finite element method, and many others. The Delaunay triangulation is so popular because the produced triangles are of extremely high quality. They are as close to equilateral as possible and elongated and narrow triangles occur very rarely. From a certain point of view, Delaunay triangulation is the optimal method for space subdivision.

Delaunay triangulation of moving points represents an extension of the static data structure for kinetic data. This enhancement allows us to utilize various features of Delaunay triangulation for number of purposes not available in the static case. For instance, the fact that the two nearest points in the triangulation will create an edge of a triangle can be used for collision detection, because the edge will certainly be created before the collision occurs (see [7]). Together with the transformation to Voronoi diagram (for details on this data structure see for instance [1, 16, 21]), the kinetic Delaunay triangulation may provide a good base structure for path planning algorithms in an environment with moving objects. Even the applications that are based on nonphysical models may use kinetic Delaunay triangulations - one such example is a kinetic triangulation based video compression (see [18]). In general, the kinetic extension of Delaunay triangulation may be found useful in any application where the data represent moving points or objects.

Several approaches for handling the movement of points in the triangulation exist. These approaches may be divided into two groups according to the actions performed upon the set of points and upon the triangulation itself. The former group contains methods that remove the moving points at their starting positions and reinserts them into the triangulation at their target coordinates. The latter group consists of methods that alter the triangulation (usually by using edge swaps) in order to modify it to the legal state.

Apart from its simplicity, the remove-reinsert approach has some significant disadvantages. Besides the fact that triangle location is necessary for the point removal (see [4]), some important events may be unwillingly skipped due to using this approach. Considering an application for collision detection, by removing a point and reinserting it back to the triangulation at a new position, we may accidentally miss a collision with another point. This will happen if the new position is relatively far away from the starting coordinates (i.e., the point is moving fast) and thus some edge swapping is missed. If one of these edges represents a collision edge between this point and another point that is moving slowly, the collision will be ignored. However, in applications, where this kind of errors is not possible or is not important, the remove-reinsert based methods may be fully sufficient.

Continuous movement of the points used in the other approach requires the triangulation to change as a reaction to the topological events that are caused by this movement in order to remain legal. To be able to do this, the triangulation structure may utilize a priority queue to keep track of the topological events. When a request for the current state of the triangulation arises, topological events are popped from the queue and processed until the time of the next topological events is greater than the requested time. A modification of this approach is presented in [6]. This solution divides time into small intervals and all topological events and collisions are postponed to the end of the interval. This may cause that collisions will be detected late and the triangulation will become illegal for most of the duration of a time interval. However, in some cases this approach may be very efficient.

This diploma thesis presents an overview of variety of methods for handling Delaunay triangulation of kinetic data and presents our own implementation of one such method that use the approach with continuous point movement and utilizes a priority queue to keep track of the topological events. Even though these approaches are relatively well known and have been described, the mathematical apparatus needed for them is often omitted in the papers and, besides its general idea, is left undescribed. In this thesis we present a new algorithm for computing the times of the topological events and compare them to some other possible computation methods.

Chapter 2 describes the Delaunay triangulation in general and provides insight into the construction methods which are used most often and some other related algorithms, such as point removal from the Delaunay triangulation. Information on kinetic data and a general description of various approaches to kinetic Delaunay triangulation are provided in Chapter 3. Some necessary polynomial features, methods for solving polynomials of the fourth or lesser degree and some useful related mathematical structures are described in Chapter 4. Our method for handling Delaunay triangulation of kinetic data is described in Chapter 5 and its performance is documented in Chapter 6. Conclusion of the whole thesis is provided in Chapter 7.

2 Triangulation and Delaunay Triangulation

2.1 Definitions

2.1.1 Triangulation

According to [15], triangulation $T(S)$ of a set of points S in the Euclidean plane is a set of edges E such that

- no two edges in E intersect at a point not in S,
- the edges in E divide the convex hull of S into triangles
- the space division of the convex hull is maximal

2.1.2 Delaunay Triangulation

Delaunay triangulation $DT(S)$ of a finite set S of n points in $2D$

$$S = \{P_1, P_2, ...P_n\}$$

is the triangulation that fulfills the condition that no point is inside the circumcircle of any triangle in $DT(S)$. This property, known as the Delaunay condition, is a key feature in our application and must be preserved over time despite the movement of the points.

In addition to the Delaunay condition, the Delaunay triangulation may also be defined as the one triangulation, from all possible triangulations of the set of points S, with the largest minimal inner angle of each triangle. This feature is often referred to as the optimality in the sense of $MaxMin$ angle criterion (see [10]).

Another possible definition of Delaunay triangulation is given in [13]. According to this definition, the Delaunay triangulation $DT(S)$ is dual structure to Voronoi diagram $Vor(S)$. Voronoi diagram is a set of all the points that are equally distant from two or more points in S and that do not lie closer to any other points in S. According to [12], Voronoi diagram in d-dimensional space may be mathematically described as.

$$Vor(S) = \{x \in E^d : \forall p_k, \exists p_i, p_j : p_i, p_j, p_k \in S; i \neq j \neq k : \|p_k - x\| \geq \|p_i - x\| = \|p_j - x\|\}$$

The mutual relationship of $Vor(S)$ and $DT(S)$ is described in Figure 2.1. The duality of these two structures may be used for construction algorithms that convert one of them into the other. However, their performance is usually worse than the performance of direct construction algorithms.

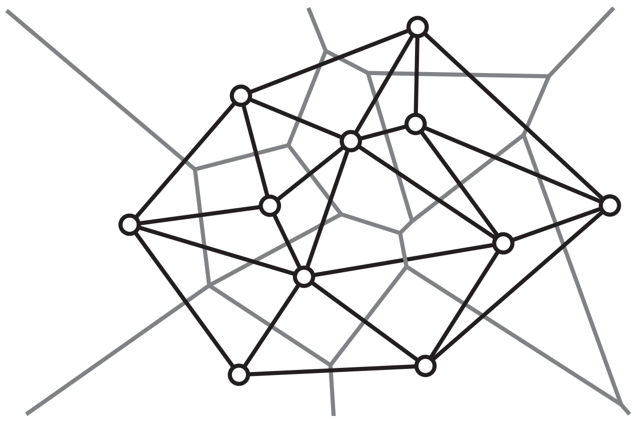


Figure 2.1: Mutual relationship between Delaunay triangulation and Voronoi diagram.

2.1.3 The Incircle Test

To determine if a triangle $P_1P_2P_3$ and a point P_4 satisfy the Delaunay condition of the empty circumcircle, the incircle test must be made over the three points of the triangle and the considered point. If $P_i = [x_i, y_i]$ where $x_i, y_i \in \mathbb{R}$ represent the coordinates of points

$P_1, ..., P_4$, then we can determine the position of P_4 against the circumcircle of the triangle $P_1P_2P_3$ according to the sign ot the determinant of the matrix $\mathbf{I}$ (for details see [10]):

$$\det \mathbf{I} = \det \begin{pmatrix} x_1 & y_1 & x_1^2 + y_1^2 & 1 \\ x_2 & y_2 & x_2^2 + y_2^2 & 1 \\ x_3 & y_3 & x_3^2 + y_3^2 & 1 \\ x_4 & y_4 & x_4^2 + y_4^2 & 1 \end{pmatrix} \tag{2.1}$$

If the vertices of the triangle $P_1P_2P_3$ are oriented counterclockwise, then the positive sign of Eq. (2.1) means that P_4 lies inside the circumcircle of $P_1P_2P_3$, negative sign means that P_4 lies outside and zero always means (independently on the orientation of the vertices of the triangle) that P_4 lies on the circumcircle. Figure 2.2 shows an example of these values.

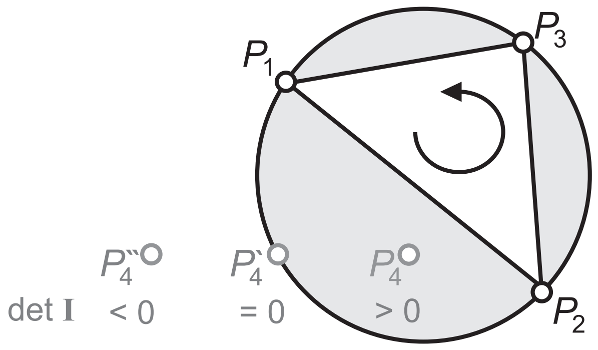


Figure 2.2: Example of some values of the incircle test.

The incircle test in 2D in fact determines whether the projection of point the P_4 on the paraboloid $z = x^2 + y^2$ (see [3]) lies above, on or below the plane defined by the projection of P_1, P_2 and P_3 on the same paraboloid. Explanation on a 1D example is given in Fig. 2.3. In this case, the test computes whether the point P_3 lies within the interval determined by points P_1 and P_2 by determining the position of its projection P_3' against the line given by the projections P_1' and P_2' of P_1 and P_2, respectively, on the parabola $y = x^2$. Further explanation of the incircle test may be found in [6].

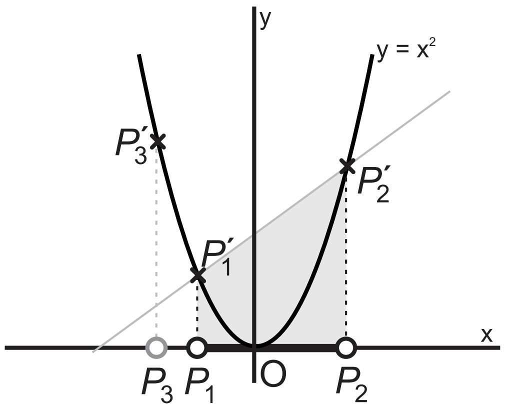


Figure 2.3: 1D example of the incircle test - the test will determine the position of the point P_3' against the line given by the projections P_1', P_2' of P_1 and P_2 on a parabola.

2.2 Delaunay Triangulation Construction Methods

2.2.1 Construction Algorithms Criteria

When evaluating a Delaunay Triangulation algorithm, several features may be considered, each with varying significancy depending on the intended target application of the algorithm. According to [2], these features include (but are not limited to):

Time and space complexity Determines the overall performance of the algorithm. Runtime needed for the algorithm is a vital feature but a small memory consumption also becomes necessary as the data sets grow larger.

Online Some algorithms do not need all the input data as one large package. Rather than processing the whole dataset in one step, the data are inserted into the triangulation one point at a time.

Extensibility to higher dimensions An algorithm may be usable only in a plane. Some of the 2D algorithms may however be extended to three or more dimensions.

Parallelism Some algorithms may be from their nature unsuitable for parallel execution. Others may be parallelized with a little effort.

The following construction methods were mostly taken from [3, 10, 12, 14], where further description of these methods may be found. Special attention should be paid to incremental insertion construction method, because it is essential for the proposed solution of the described problem. It is thus described apart from other methods in Chapter 2.3.

2.2.2 Local Optimization Algorithm

This simple algorithm is based on the fact that the edges of Delaunay triangulation are locally optimal. Thus any initial triangulation may be modified into Delaunay triangulation by altering the edges in the sense of optimization according to the $MaxMin$ angle criterion. According to [10, 12], the edge e will be replaced by the edge e' (see below) if and only if:

- e does not lie on $CH(P)$.
- The quadruple $P_1P_2P_3P_4$ created by the two triangles adjacent to e is convex.
- Swapping e for the other possible inner edge e' of $P_1P_2P_3P_4$ will increase the angle of the minimum inner edge of $P_1P_2P_3P_4$.

This method, however simple and relatively stable, is not often used because it may not be reliably extended to three dimensions, it is not online and it is difficult to be parallelized. The performance of this method also depends strongly on the time complexity of the method used to create the initial triangulation, with the worst case complexity being $O(n^2)$ and the expected complexity $O(n)$.

2.2.3 Incremental Construction Algorithm

Given a set of points, the incremental construction algorithm chooses one point at random and its nearest neighbor, creating the first edge of the triangulation. For each of the edges in the triangulation, such point P_i is found that the circumcircle of the triangle created by the edge and P_i is minimal.

This construction method thus uses the empty circumcircle condition of Delaunay triangulation (or empty circumsphere in three dimensions, where the algorithm is similar).

Even though the algorithm is very simple, it is not usually used, because its performance is strongly influenced by the methods used for finding the points with the minimal circumcircle diameter. In 2D, the expected time complexity is $O(n \log n)$, the worst case time complexity is $O(n^2)$, the algorithm is not online, may not be parallelized and may be extended to 3D. In three dimensions, the situation is even worse, because, according to [12], without any acceleration, the worst time complexity in three dimensions is equal to $O(n^3)$. Another downside of this algorithm is its lack of stability. Triangle overlaps may occur which results in an illegal data structure (as defined in Chapter 2.1.1, no two edges of a triangulation may intersect).

2.2.4 Construction via Extension to Higher Dimension

As described in [4], there is a well known relationship between Delaunay triangulation in d-dimensional space and a convex hull in *d+1*-dimensional space. The original points are projected on a *d+1*-dimensional paraboloid in a way very similar to the process described in Chapter 2.1.3 and the convex hull of their projections is constructed and then projected back into the original d-dimensional space.

Because it is rather complicated, this method of constructing Delaunay triangulation is used very rarely and its properties are derived from the algorithm for the convex hull creation.

2.2.5 Divide and Conquer Algorithm

Divide and conquer (D&C) represents a principle rather than a single method. This approach is widely used in computational geometry and many other (even completely unrelated) fields. When constructing the Delaunay triangulation, the point set is recursively divided and the parts are triangulated separately and then connected together. The triangulation itself is done by any sufficient algorithm (one of the mentioned here or some other).

Even though the implementation of this type of algorithm is rather complicated, it has been proven it is time optimal in for the worst case. Due to some complications with sorting in higher dimensions, the extension of this algorithm to dimensions higher than two is nontrivial (see [2]), it is not online, but it may be parallelized with relatively little effort.

2.2.6 Incremental Insertion Algorithm

This algorithm will be described later in Chapter 2.3 in detail, this section should provide a comparison to the other algorithms and provide a general idea of its function.

At the start of the algorithm, a sufficiently large triangle is created, which contains all the points $P_1, ..., P_n$. For each of these points, a triangle containing that point is located and divided into three triangles by connecting the point to its vertices. If the point lies on an edge, then the two adjacent triangles are split into four. The newly created edges are then tested for the Delaunay condition and swapped if necessary. After all the points are inserted, all the edges that contain at least one of the vertices of the large triangle added at the start of the insertion are removed.

This construction method is online, may be parallelized and extended to 3D. Its time and memory complexity is derived from the used point location technique (see further).

2.3 Delaunay Triangulation Construction with Incremental Insertion

2.3.1 Initial Simplex Construction

In order to be able to locate the triangle containing each of the points in P, a sufficiently large triangle must be created before the insertion starts[1]. This initial triangle must be large enough to contain all the points to be inserted, but must not be too large, because it would negatively alter the numerical stability of the algorithm. The ideal size of the

[1]In three dimensions, a tetrahedron will be created. In general a d-dimensional simplex is created, but this thesis focuses on 2D case, so only the triangles will be considered.

triangle in 2D is considered to be $(K, 0)$, $(0, K)$ and $(-K, -K)$ where K is equal to a multiple of the size of the longer side of the rectangle containing the whole triangulation area (see [12]). Note that different values of K may be used and will work well, the general idea mentioned before should be kept in mind though. For instance [26] proposes the value of K to be 10 times the size of the bounding box or even larger. Figure 2.4 illustrates the idea of the initial bounding triangle.

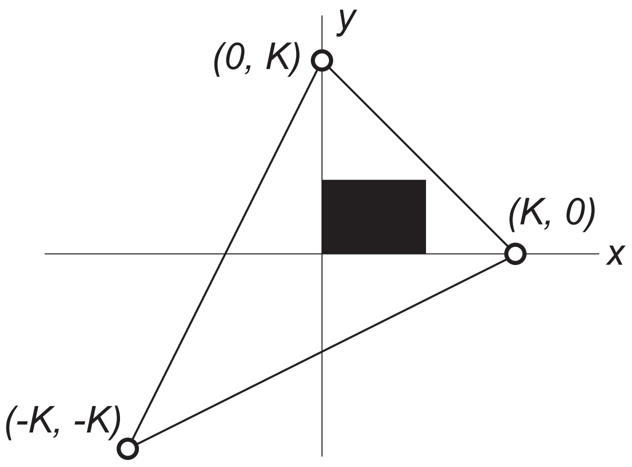


Figure 2.4: A triangle that contains the whole triangulation area is added at the start of the incremental insertion algorithm.

2.3.2 Point Location

Two general approaches to point location techniques are most widespread and the performance of the incremental insertion algorithm strongly depends on the used point location technique. The more complicated one uses sophisticated data structures (such as DAG[2], skip-list and others - see [3, 14]) and usually achieves better results in the field of time complexity and overall performance (in 2D we can expect time complexity of $O(\log n)$ per point and memory complexity of $O(n)$ where n is the total amount of points in the triangulation). On the other hand, these structures are not very well suited for time-dependent triangulations because changes in the tree-like (or even more complicated) hierarchies are

[2]Directed acyclic graph

nontrivial. For this reason, it will not be described. Details on these data structures may be found in [3, 12, 14] and many others.

The walking algorithms represent an easy-to-implement and very popular way of point location. Because they do not depend on any additional data structures, the modification of the triangulation as a result of the movement does not represent a problem. The principle of walking algorithms is that the triangle containing the searched point is found by visiting a random triangle at first and then searching for which of the neighbors of the currently visited triangle is nearest to the searched point. The search may be performed in various different ways, which may be found for instance in [14, 22]. Depending on the strategy for searching for the first random triangle and the used walking strategy, the point location may vary from $O(n^{1/2})$ to $O(n^{1/3})$ per point location in 2D case.

2.3.3 Point Insertion and Edge Legalization

When a point is inserted into the triangulation, two cases may occur in 2D. It is either inserted into an interior of a triangle or on an edge (we will further ignore the singular case, where two points are identical and thus the inserted point lies on a vertex of the found triangle). Figure 2.5 shows both of these cases (with P_r being the inserted point).

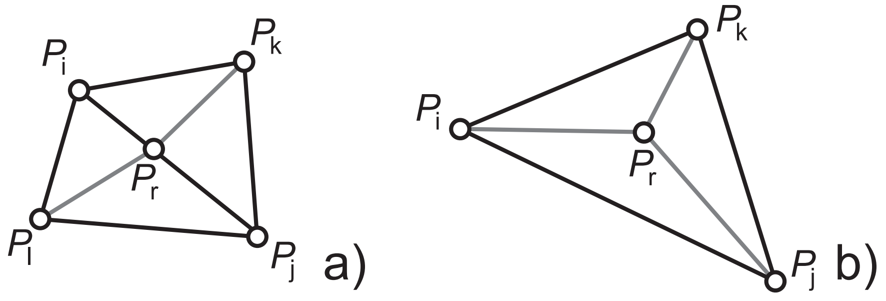


Figure 2.5: Triangle splitting in the creation of Delaunay Triangulation via incremental insertion. Case a) shows the situation when the inserted point lies on an edge common to two triangles, case b) shows the way of splitting a single triangle that contains the inserted point.

There exist more cases in 3D, which are generally more complicated to handle. These cases include the point being inserted on an edge or face of a tetrahedron and are described for instance in [11].

As displayed in Figure 2.5, if the point is being inserted into a single triangle, this triangle is split into three triangles by connecting the newly inserted point to each of its three vertices, thus creating triangles $P_iP_jP_r$, $P_jP_kP_r$ and $P_kP_iP_r$. If the point lies on an edge adjacent to two triangles, then it is confected to their vertices which are opposite to the edge that contains the inserted point. Note that the inserted point may never lie on an edge of the convex hull of the triangulation structure, because the initial triangle created at the start of the algorithm contains the whole triangulation area and is thus equal to the convex hull of the triangulation.

The newly created edges may not be locally legal in the sense of Delaunay condition, so a legalization step is conduced upon them, which is based on a very similar idea as the Local optimization algorithm (see Chapter 2.2.2 above).

2.3.4 Algorithms

The two algorithms below show the way of creating Delaunay triangulation by incremental insertion without the point location techniques. Alg. 2.1 shows the point insertion and Alg. 2.2 describes the legalization of newly created edges. Algorithms were taken from [3], where additional description may be found.

Algorithm 2.1: Creation of Delaunay triangulation with incremental insertion of points.

```
Input:
  • $P = \{P_1, ..., P_n\}$ set of $n$ points in 2D.
  • $O = \langle x_{min}, x_{max}\rangle \times \langle y_{min}, y_{max}$ a subset of the Euclidean plane that contains all the
    points in $P$.
Output: Delaunay triangulation $DT(P)$ of the points in $P$
Create $P_{-1}P_{-2}P_{-3}$ - a triangle that encapsulates the whole triangulation area $O$.
Initialize $DT(P)$ as the triangulation consisting of the single triangle $P_{-1}P_{-2}P_{-3}$.
foreach $P_r \in P$ do
    Find a triangle $T = P_iP_jP_k \in DT$ that contains the point $P_i$.
    if $P_r$ lies on a vertex of $T$ then
        Discard $P_r$;                     // Two identical points in the set.
    else if $P_i$ lies on an edge $E$ of $T$ then
        Let $E = P_iP_j$ as in Fig. 2.5a.
        Let the other triangle adjacent to $E$ be $P_jP_iP_l$ as in Fig. 2.5a.
        Split both triangles incident to $E$ thereby creating four new triangles.
        LegalizeEdge($P_r$, $P_iP_l$, $DT(P)$)
        LegalizeEdge($P_r$, $P_lP_j$, $DT(P)$)
        LegalizeEdge($P_r$, $P_jP_k$, $DT(P)$)
        LegalizeEdge($P_r$, $P_kP_i$, $DT(P)$)
    else
        Split the triangle $T$ as in Figure 2.5b into three new triangles.
        LegalizeEdge($P_r$, $P_iP_j$, $DT(P)$)
        LegalizeEdge($P_r$, $P_jP_k$, $DT(P)$)
        LegalizeEdge($P_r$, $P_kP_i$, $DT(P)$)
    end
end
Discard $P_{-1}$, $P_{-2}$, and $P_{-3}$ with all their incident edges from $DT(P)$.
Return $DT(P)$.
```

Algorithm 2.2: Edge legalization for the construction of DT.

```
Input:
  • $P_r$ - a point being inserted into $DT(P)$.
  • $P_iP_j$ - the edge of $DT(P)$ that may need to be flipped.
  • $DT(P)$ - current state of the Delaunay Triangulation.

if $P_iP_j$ is illegal then
    Let $P_iP_jP_k$ be the triangle adjacent to $P_rP_iP_j$ along $P_iP_j$.
    Replace $P_iP_j$ with $P_rP_k$.;                          // Flip $P_iP_j$
    LegalizeEdge($P_r$, $P_iP_k$, $DT(P)$)
    LegalizeEdge($P_r$, $P_kP_j$, $DT(P)$)
end
```

2.4 Point Removal in Delaunay Triangulation

Due to the fact that some of the algorithms for the kinetic Delaunay triangulation directly use a removal of points from the triangulation, and other may take advantage of this possibility, it will be useful to describe one version of this algorithm (proposed by Devillers in [4]).

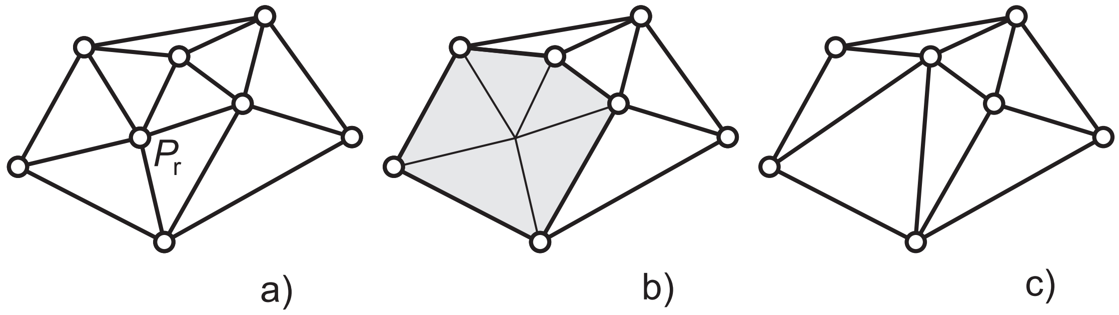


Figure 2.6: Removing point P from the triangulation.

Let us have Delaunay triangulation $DT(P)$. If we want to remove the point P_r from the triangulation, we have to remove all the triangles sharing this point as a vertex. By removing these triangles, a hole is created in the triangulation. This hole defines a star-shaped polygon, which has to be retriangulated (the changes in the triangulation are strictly local and limited to this polygon, no other triangles in the triangulation will be affected by the point deletion). The whole process is illustrated in Figure 2.6.

The only question is how to retriangulate the hole. Let us now define an ear of a polygon $Q = \{Q_1, Q_2, ..., Q_n\}$: any three consecutive vertices $Q_iQ_{i+1}Q_{i+2}$ of Q form an ear of Q, if the line segment Q_iQ_{i+2} is inside Q. According to [4], an ear of Q will be said Delaunay if the circumcircle of triangle $Q_iQ_{i+1}Q_{i+2}$ does not contain any other vertices of Q.

Devillers further assigns a priority (or power) function to each ear as in Eq. 2.2, and shows, that by cutting the ear with the minimal priority value and adding it into the triangulation as a new triangle, we can retriangulate the hole in Delaunay sense.

$$power(P, Q_0, Q_1, Q_2) = \begin{cases} \infty & Q_0Q_2Q_3 \text{ is oriented clockwise} \\ \frac{inCircle(P,Q_0,Q_1,Q_2)}{orientation(Q_0,Q_1,Q_2))} & \text{otherwise} \end{cases} \tag{2.2}$$

where $inCircle(P, Q_0, Q_1, Q_2)$ is the standard incircle test function as defined in Chapter 2.1.3 and $orientation(Q_0, Q_1, Q_2))$ is another determinant test function, defined as in Eq. 2.3.

$$orientation(X_1, X_2, X_3) = \det \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix} \tag{2.3}$$

Where $X_i = (x_i, y_i), i \in \{1, 2, 3\}$ are the points to be tested.

Using this test, we are able to determine if the three tested points lie on a line, are oriented clockwise or are oriented counterclockwise. Details of the test and the priority function may be found in [4].

3 DT of Kinetic Data

3.1 Definitions

3.1.1 Point Movement

Given a set of points $P = \{P_1, ..., P_n\}$ as in Chapter 2.1.2 and Delaunay triangulation $DT(P)$ created from these points, let us state that the points move along linear trajectories with constant velocity vectors:

$$P_i(t) = [x_i(t), y_i(t)] \tag{3.1}$$

$$x_i(t) = x_{i0} + v_{xi} \cdot t, y_i(t) = y_{i0} + v_{yi} \cdot t \tag{3.2}$$

where $t \geq 0$, $P_i(0) = [x_{i0}, y_{i0}]$ is the initial position of the point P_i, i.e. the coordinates at which this point started it movement at the time t_0 and $v_{xi},\ vyi\ \in \mathbb{R}$ are the components of the velocity vector $\mathbf{v_i}$ adherent to the point P_i.

Let us also define a bounding area O as a subset of the Euclidean plane:

$$O = \langle x_{min}, x_{max} \rangle \times \langle y_{min}, y_{max} \rangle$$

where $x_{min}, x_{max}, y_{min}, y_{max} \in \mathbb{R}$. Let us further state that the initial position of each point must lie within the bounding rectangle O and that no point may ever leave the bounding area.

3.1.2 Priority Queue

A priority queue is an abstract data type, which provides the following operations:

- Push (i, p): add the item i to the queue with respect to the priority p.
- Pop: remove the item i with the highest priority from the queue and return it.
- And sometimes others, such as returning the first element in the queue without removing it (known as the "Head" function).

The items in the priority queue may be sorted (in this case, the Push operation usually represents some form of the Insert Sort algorithm with time complexity $O(\log n)$ and the Pop operation simply removes and returns the first element of the queue with time complexity $O(1)$). The other common approach is to insert the items into the queue without any form of sorting (time complexity $O(1)$). In this case, the Pop operation must search the whole queue for the item with the maximum priority value (time complexity $O(n)$).

3.1.3 Topological Events

Topological event is a time instant when four points that create two adjacent triangles in Delaunay triangulation become cocircular due to their movement. Result of this special point configuration is a topological change in the triangulation structure performed in order to keep the structure legal. This topological change is represented by an edge swap - replacing the two adjacent triangles with the other possible triangle configuration of the four cocircular points.

3.2 Discrete Time Approach

3.2.1 Basic Concept

The ability of Delaunay triangulation to remove or insert the points from or into the triangulation at request is often referred to as the fully dynamic property. As mentioned in [16] this feature is rather used to manage time dependent datasets, where some points are used only for a limited time duration, than to simulate point movement.

An obvious advantage of this approach is the fact that (when used to simulate point movement) the time complexity is independent on the trajectories of the moving points. The algorithms for removing points and adding points from and into the triangulation are "unaware" of the trajectories and thus are not affected by them.

On the other hand,the main disadvantage of this approach lies in the discrete understanding of time, which makes it unsuitable for quite a large set of applications. As illustrated in Figure 3.1, the two consecutive iterations of the discrete time algorithm may cause that a temporal triangle configuration is missed which may lead to some errors (for instance in a collision detection application). Figure 3.1a and 3.1c show the triangulation before and after the remove and reinsert step, while Figure 3.1b shows the missed event that occurred inside the discrete time step.

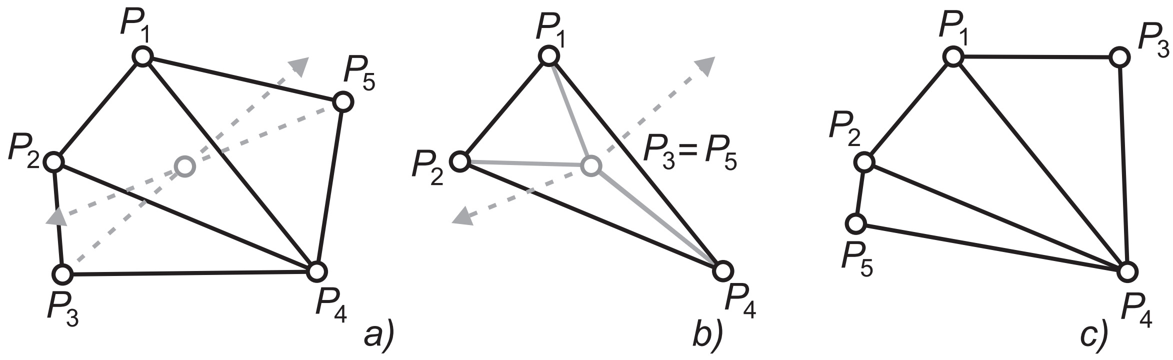


Figure 3.1: The discrete understanding of time may lead to missing some triangle configurations.

3.2.2 Practical Use

The practical use of the remove-reinsert approach is described in [16], where the general idea may be found. According to this paper, it may be used for a shape generalization (as in the curve smoothing) to simulate the movement of the points towards a curve. This use is described in [24]. The mentioned paper - [16] - further states that this approach may even be used (with some modifications) for collision detection - see [8].

3.3 Continuous Movement Approach

3.3.1 Overview

The ability to control Delaunay triangulation in a continuous time possesses few very significant advantages for the price of slightly more complicated algorithms. The first and the most important of them is the fact that the structural data are accessible for each time instant of the program lifecycle and thus are usable for a wider variety of applications, such as a collision detection, without any serious modifications.

The following section describes the basic ideas of the most often discussed algorithms for managing the kinetic Delaunay triangulation structures.

3.3.2 Simplex Flipping

Even although the following algorithm (described in [21]) is originally a part of point removal algorithm for Delaunay triangulation, it is based on an idea of continuous movement. Despite the fact that it only allows the movement of one point at a time, perhaps this idea could be evolved into such a state that would allow a full-scale continuous movement.

The basic thought of the algorithm is the following - an illegal kinetic data structure[1] may be repaired as long as the movement of the points does not cause an overlapping of the edges of this structure. A sample illustration is given in Figure 3.2 - the case a) represent the "maximally illegal" structure that is allowed (point P_4' is the furthest position of P_4 if it moves along the displayed trajectory) and the case b) shows an unallowed state (the triangle $P_1P_4P_2$ overlaps the triangle $P_1P_2P_3$). Note that neither of the moved cases displays a legal Delaunay triangulation.

The times when the triangulation reaches the maximum allowed deformation (and thus needs to be repaired) are computed by using a modification of the orientation test (see Eq. 2.3). Similarly to the incircle test (see Eq. 2.1), the singular case of the orientation test is recognized by a zero value of the determinant. This singularity is represented by

[1]i.e., a triangulation that does not fulfill the Delaunay condition, or corresponding "Voronoi" tesselation.

three colinear points in 2D (or similar configurations in higher dimensions) and defines the maximum allowed deformation of the point structure - i.e. the maximum value of time when the orientation of the simplex is the same (the determinant value has the same sign) as it was at the beginning of the movement.

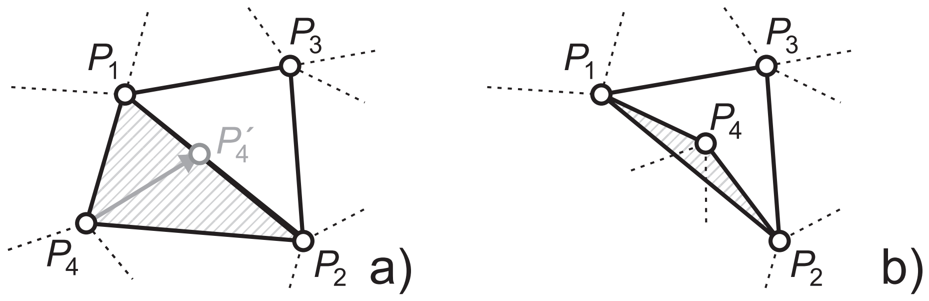


Figure 3.2: The maximum allowed deformation of the triangulation and the unallowed state.

In order to find the times when the triangulation reaches the singular case, the following equation must be computed (defined in [21])[2]:

$$0 = \nu_0 + \lambda_i \nu_\Delta \tag{3.3}$$

where ν_0 is given by Eq. 3.4 at the start of the movement, λ_i is the maximum allowed duration of the movement along the trajectory defined by velocity vector stored in ν_Δ.

$$\nu_0 = \det\begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{pmatrix} = \det\begin{pmatrix} x_1 - x_2 & x_2 - x_3 & x_2 - x_4 \\ y_1 - y_2 & y_2 - y_3 & y_2 - y_4 \\ z_1 - z_2 & z_2 - z_3 & z_2 - z_4 \end{pmatrix} \tag{3.4}$$

$$\nu_\Delta = \det\begin{pmatrix} \Delta_x & x_2 - x_3 & x_2 - x_4 \\ \Delta_y & y_2 - y_3 & y_2 - y_4 \\ \Delta_z & z_2 - z_3 & z_2 - z_4 \end{pmatrix}$$

[2]As mentioned before, there is only one point moving in the triangulation. Let us state that the moving point is $P_1 = [x_1, y_1, z_1]$ and that its velocity vector is defined as $\Delta = [\Delta_x, \Delta_y, \Delta_z]$.

By solving these equations for each adjacent simplex pair and finding the shortest legal time interval for each point, one will be able to compute the periods of triangulation legality.

Each time the triangulation reaches the singular state, it is repaired by using the simplex flips. In 2D, these flips are represented by simple edge swaps - the current triangle configuration of four points is swapped to the other possible configuration. In higher dimensions, the situation becomes little more complicated and a wider variety of possible simplex configuration exist. This problem is discussed in [21], where the details may be found.

3.3.3 Postponed Event Approach

This approach (described in [6]) represents a special kind of a hybrid method and could be understood as an extension of the previous Simplex Flipping algorithm (see Chapter 3.3.2). It is based on continuous movement principles as described in [1, 7, 8] and others, but exploits the discretization of time and allows the triangulation to reach such states that would be normally illegal. It is strongly dependent on the intended target application and may be unsuitable for some problems.

The postponed event approach introduces two types of events - cocircular events (as described earlier in Chapter 3.1.3) and collision events between two spheres (the underlying data structure is the Delaunay triangulation of spheres and the vertices thus represent these spheres). However, these events are not executed immediately when the inner time of the triangulation matches their time. The lifecycle of the triangulation is rather divided into the intervals T_i:

$$\begin{aligned} T_i &= \langle t_i, t_i + h_i \rangle \\ t_{i+1} &= t_i + h_i \end{aligned}$$

where $t_i \in \mathbb{R}$ is the beginning time of interval T_i and $h_i > 0$ is so called time horizon -

i.e., the length of the interval T_i, with t_0 being the start time of the triangulation and the configuration of the triangulation, which is legal in the Delaunay sense at this time.

The events scheduled for execution during the current interval are stored and postponed until the end of the interval, when they are executed using simplex flips for topological events and velocity vector modifications for collision events. The only problem which remains to be solved is the time division. The durations of the intervals should be as long as possible, but - on the other hand - the triangulation must not be in an illegal state at the end of any of them. The Delaunay condition may (and most probably will) be broken as a result of the movement, but (similarly to the previous algorithm), overlapping of two adjacent simplices is not allowed (see Figure 3.2).

As a result of the event postponing, some collisions may be detected later than they occur (possibly at the end of a time interval), it is up to the application if this fact represents some sort of a threat or a serious inconvenience. The mentioned thesis uses the kinetic Delaunay triangulation for collision detection among the spherical metal grains. In this case, certain small amount of imprecision is acceptable and the simulated grain overlapping does not influence the computation too negatively to make this method unusable. However, the time horizons must be small enough to prevent a total merging of the two colliding grains.

3.3.4 Continually Legalized Triangulation Approach

This approach, described in [1, 7] uses a priority queue to store topological events (other types of events may be used, but only the topological ones are necessary). These events are sorted by their computed times and are executed as the inner time of the triangulation reaches the precomputed event values.

Because this approach represents the base idea of our algorithm, it will be described in detail in the following chapter.

3.4 Continual Legalization Algorithm

3.4.1 Overall Functionality

As described in [1, 7] the lifecycle of the algorithm may be divided into two phases (not counting the construction of the triangulation itself, which does not differ from the static case). The first of them - the initialization phase - determines the nearest topological event for each pair of adjacent triangles in the triangulation by computing the time their four points become cocircular. If the kinetic system is used for a collision detection, the collision times are also determined, because they alter the points as well. This includes both the point-point collision and the point-wall collision (let the boundaries of the triangulation area be denoted as the walls). If some other time and structure dependent events are needed for any other purposes, their times are also determined in the initialization step.

All the computed events are then placed into a priority queue with the priority defined in such a way so that it ensures that the events taking place earlier will be popped from the queue before the events scheduled after them. One such a priority function may be defined as follows (assuming the definition of the priority queue as described in Chapter 3.1.2):

$$p = -t_{event} \tag{3.5}$$

where $t_{event} \in \mathbb{R}$ is the time of the event. This priority function, along with the fact that the item with the maximum priority value is popped from the queue first, ensure the required functionality.

After the initialization step, the lifecycle of the algorithm consists of repeating the iteration step each time the triangulation state has to be updated. This step consists solely of popping the events from the top of the priority queue, executing them (thus changing the topological structure of the triangulation or some points features) and pushing new events into the queue. The pop-execute-push cycle is repeated until the time of the event on the top of the queue (i.e., the nearest future event) is greater than the time of the triangulation. The overall functionality of the algorithm is summarized in Alg. 3.1.

Algorithm 3.1: Overall functionality of the Continual Legalization algorithm.

```
Input:
  • $Q$ - Priority queue
  • $DT(P)$ - Delaunay triangulation of kinetic data
Output:
  • Continually legalized Delaunay triangulation for kinetic data.
Auxiliary:
  • $t_{curr}$ - The current time of the triangulation
  • $Ev$ - temporary event variable

// Initialization step
foreach Adjacent triangle pair $T_i$, $T_j$ in $DT(P)$ do
    Compute the next future topological event $Ev_{ij}$ at time $t_{ij}$
    if $Ev_{ij}$ exists then
        $Q.push(Ev_{ij}, t_{ij})$
    end
end

// Iteration step
while Time of $Q.head() > t_{curr}$ do
    $Ev \leftarrow Q.pop()$
    Execute $Ev$
    Push new events into $Q$ as required
end
/* This step is repeated as required during the whole lifecycle for
   increasing values of $t_{curr}$                                    */
```

3.4.2 Explanation of the Topological Events

If the triangulation contains at least one moving point with a nonzero velocity vector, its structure will have to change in time due to the Delaunay condition. As shown in [1, 7] for some time the moving points may move without any topological changes, but as soon as one of them enters the circumcircle of a nearby triangle, the triangulation becomes illegal and a topological change takes place. Example of this event is given in Figure 3.3.

In this example the triangles $P_1P_2P_3$ and $P_1P_4P_2$ represent a Delaunay valid configuration. As the point $P_4 \rightarrow P_4' \rightarrow P_4''$ moves towards the circumcircle of the triangle $P_1P_2P_3$

and enters it, the singular case occurs. Because of the fact that both available triangle configurations of four cocircular points are Delaunay-legal, the triangulation is formally valid until the moving point enters the interior of the circumcircle of the triangle. At this point the Delaunay condition becomes violated and the triangulation must be repaired by processing the topological event.

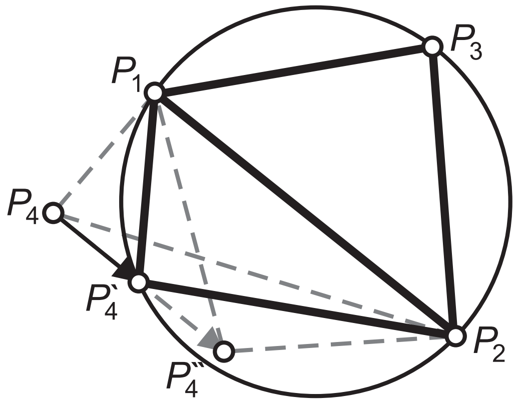


Figure 3.3: Triggering of a topological event.

3.4.3 Obtaining the Topological Events

The topological events are caused by a time-dependent point movement and determined by the time when four points become cocircular. In order to compute these times, a modified incircle test has to be performed:

$$\det \mathbf{I}(t) = 0 \tag{3.6}$$

where $\mathbf{I}(t)$ is time-dependent incircle test matrix (see Eq. 2.1):

$$\mathbf{I} = \begin{pmatrix} x_1(t) & y_1(t) & x_1^2(t) + y_1^2(t) & 1 \\ x_2(t) & y_2(t) & x_2^2(t) + y_2^2(t) & 1 \\ x_3(t) & y_3(t) & x_3^2(t) + y_3^2(t) & 1 \\ x_4(t) & y_4(t) & x_4^2(t) + y_4^2(t) & 1 \end{pmatrix} \tag{3.7}$$

where $x_i(t), y_i(t); i = 1, ..., 4$ are the time-dependent coordinates of the moving points $P_1, ..., P_4$ upon which is the test performed (see Chapter 3.1.1).

In order to compute this equation and thus obtain the times of the topological events for the four considered points, a polynomial of the fourth or lesser degree has to be solved.

3.4.4 Lifecycle of a Topological Event

As mentioned before, the topological events are created during the initialization step of the algorithm along with other types of events (if they are relevant). For each edge e in the triangulation, its two vertices are tested for a mutual collision (even if the triangulation is not intended as a data structure for collision detection - see further) and if e is shared by two triangles, their four points are tested for future topological events. If any such events exist for these four points, the nearest of them is added into the priority queue. Other future topological events are discarded because the triangulation will change before their execution. Topological events in the past (which may be a byproduct of this computation) are also discarded for obvious reasons.

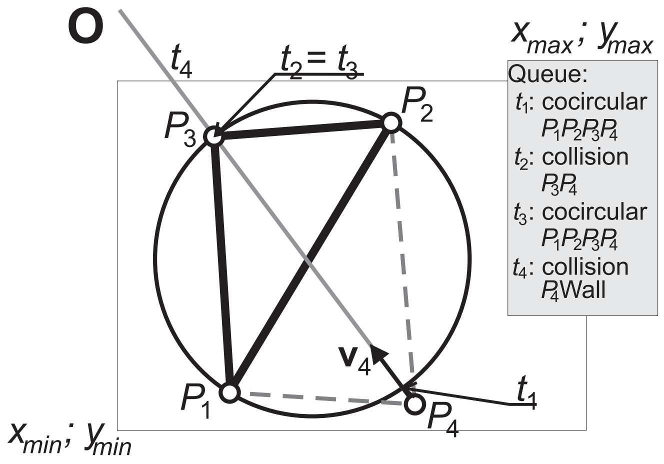

O
t_4
$t_2 = t_3$
$x_{max}; y_{max}$
Queue:
t_1: cocircular $P_1P_2P_3P_4$
t_2: collision P_3P_4
t_3: cocircular $P_1P_2P_3P_4$
t_4: collision P_4Wall
P_2
P_3
$\mathbf{v_4}$
P_1
P_4
t_1
$x_{min}; y_{min}$

Figure 3.4: An example of queueing events of several types.

Events of all types are then stored in a single priority queue as shown in Figure 3.4. In this figure we can see that the point P_4 moves with the velocity vector $\mathbf{v_4}$ while all other points are motionless. The movement of this point will cause scheduling of the events displayed in the queue box of this figure. The fact that not only the nearest future events are stored in the queue in this case may be confusing, but the figure serves only as an example of the contents of the queue.

Note that some of the events in the figure are of special significancy. The collision event of P_4 with the boundary of the triangulation area **O** that takes place at the time t_4 determines the time when P_4 leaves this area. A proper handling of this event and of all the other events of this kind is necessary to keep all the points within the bounding area. Another important instant is represented by two events - the collision event of the points P_3 and P_4 and the second scheduled topological event. In this case the collision event between two points represents a Delaunay condition singularity. When the positions of any two points become identical, this pair of points is cocircular with any two other points. Situations such as this one are unwanted and represent a threat to the triangulation structure because of a high probability of numerical errors. Some proper ways of handling of this type of singular cases exist and will be discussed later in Chapter 5.

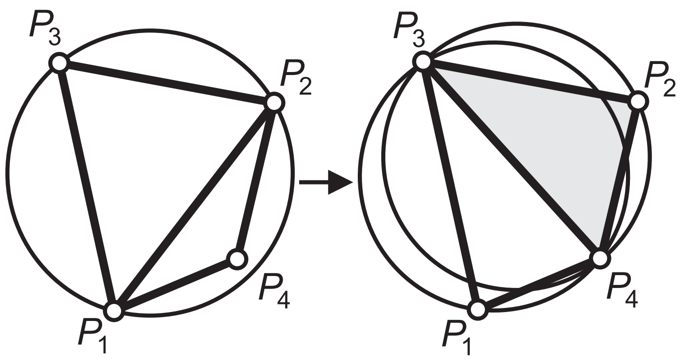


Figure 3.5: Edge swap as a result of a topological event.

When a topological event is popped from the queue and triggered, it causes local changes in the triangulation (the fact that the changes will be local is shown in [1, 7]). The process of triggering a topological event of two adjacent triangles consists of swapping their common edge as shown in Figure 3.5 and scheduling new events. The new triangles and all their neighbors must be tested for new topological events, the vertices of the new edge must be tested for a mutual collision, etc. Also removing triangles from the triangulation makes some events in the queue invalid because at least one of the triangles of those events does not exist. These events must either be removed from the event queue immediately or marked in some way as invalid and discarded when popped from the queue for processing. Alg. 3.2 shows the complete procession of a topological event.

Algorithm 3.2: Processing of a topological event

```
Input: $Ev$ - the topological event on top of the priority queue; $Ev.T_1$, $Ev.T_2$ - the
       involved triangles.
Output: Update of the topology of the triangulation and the queued events.
Auxiliary:
    • $Q$ - priority queue
    • $DT$ - Delaunay triangulation of the points $P_1, ..., P_n$

$Ev \leftarrow Q.pop()$
Let $Ev.T_1 = P_1P_2P_3$ and $Ev.T_2 = P_1P_4P_2$ as in Figure 3.5
if $Ev.T_1$ is invalid or $Ev.T_2$ is invalid then
    Discard $Ev$ and exit.
end
Swap the common edge of $Ev.T_1$ and $Ev.T_2 \rightarrow Ev.T_1 = P_1P_4P_3; Ev.T_2 = P_2P_3P_4$.
if a point collision $Col$ at time $t_{Col}$ exists between $P_3$ and $P_4$
then
    $Q.push(Col, t_{Col})$
end
foreach triangle $N$ sharing a common edge with $Ev.T_1$ do
    if a nearest future topological event $Ev_1$ at time $t_{Ev1}$ exists between $Ev.T_1$ and $N$
    then
        $Q.push(Ev_1, t_{Ev1})$
    end
end
foreach triangle $N(N \neq T_1)$ sharing a common edge with $Ev.T_2$ do
    if a nearest future topological event $Ev_2$ at time $t_{Ev2}$ exists between $Ev.T_2$ and $N$
    then
        $Q.push(Ev_2, t_{Ev2})$
    end
end
```

3.5 Theoretical Bounds of Kinetic Delaunay Triangulations

The executions of topological events and rescheduling the new ones represent the vast majority of activities that are performed during the iteration step of the algorithm. It is therefore logical to attempt to discover the theoretical bounds of the number of these event executions. This work has been done and is documented in [1].

According to this paper which uses basically the same algorithm as described here, the tests for 2D case show that the number of topological events for points moving along linear trajectories grows with $\Theta(n^{3/2})$ in an average case. The paper also presents an estimation of the worst-case bounds for both 2D case and higher dimension cases, but the estimations are only for executed events, the scheduled but discarded events are not considered. Additional information on this topic may be found in [1].

4 Polynomial Solving

4.1 Definitions

4.1.1 Polynomial

Polynomial represents a special case of nonlinear equation:

$$f(z) \equiv a_n z^n + a_{n-1} z^{n-1} + ... + a_1 z + a_0 = 0 \tag{4.1}$$

where $a_n, ..., a_0 \in \mathbb{R}, a_n \neq 0$ are the real coefficients of the polynomial and z is either complex or real variable, depending on the type of the polynomial. From now on, for our purposes, we will consider $z \in \mathbb{R}$, thus making the polynomial in Eq. 4.1 a real polynomial of n-th degree.

4.1.2 Monic Polynomial

Monic polynomial $f_{mon}(z)$ may be created from any polynomial $f(z)$ (defined as above) by simply dividing all its coefficients by the value a_n, thus gaining:

$$f_{mon}(z) = z^n + a'_{n-1} z^{n-1} + ... + a'_1 z + a'_0 = 0$$

where $a'_i = \frac{a_i}{a_n}, i = 0, ..., n-1$.

4.1.3 Polynomial Roots and their Multiplicity

Let $f(z)$ be a real polynomial as defined above. A real number z_0 is called a root of multiplicity k of $f(z)$ if there is a polynomial $s(z)$ such that:

$$\begin{aligned} s(z_0) &\neq 0 \\ f(z) &= (z - z_0)^k s(z) \end{aligned}$$

If $k = 1$, then z_0 is called a simple root.

4.2 Important Polynomial Features

4.2.1 Restricting the Root Location

As proved in [9], for each root z of a polynomial $f(z)$ as defined above, the value of this root may be restricted as follows:

$$|z| \leq \left\{ 1, \frac{1}{|a_n|} \sum_{i=0}^{n} |a_i| \right\} \tag{4.2}$$

where $a_n, ..., a_0$ are the coefficients of the polynomial.

This restriction may be very useful for a certain sort of numerical methods for solving the polynomials, because it narrows the interval, which has to be searched by these methods during an attempt to enumerate the root positions.

4.2.2 Polynomial Root Count

As a result of the fundamental theorem of algebra (see for instance [28]) applied to polynomials with real coefficients, one can state that the number of the complex roots of such a polynomial will be either zero or even.

4.2.3 Polynomial Root Decomposition

A consequence of Eqs. 4.2 is the fact that each polynomial $p(x)$ may be rewritten as

$$p(x) = \sum_{i=0}^{\deg p} (x - x_i)$$

where x_i are the roots (real or complex) of $p(x)$.

This fact allows us to divide any polynomial with one known root x_1 of multiplicity r by a polynomial $(x - x_1)^r$ (thus decreasing its degree by the value of r) and continue with any ongoing computational method to discover the remaining real roots of $p(x)$ if they exist.

4.3 Analytical Methods for Solving Polynomials

4.3.1 Introduction

Analytical methods for polynomial solving are well known and often successfully used especially for polynomials of lower degrees (up to the second degree). They are based on the properties and features of the solved functions. In theory, the analytical approach is better than most of the numerical methods, because it leads to an exact result, but due to the limited precision, the result may be quite imprecise without the ability to improve it by using multiple iterations of the algorithm.

4.3.2 Analytical Formulas

Linear equation The only single root of the linear equation, which defined as

$$a_1 z + a_0 = 0$$

may be found using the following formula:

$$z = -\frac{a_0}{a_1} \tag{4.3}$$

Quadratic equation A quadratic equation may have either zero or two real roots (which may be equal, thus forming a single multiple root). The number of these roots may be determined by the value of the discriminant D of the polynomial. If we define the quadratic equation as:

$$a_2 z^2 + a_1 z + a_0 = 0$$

then the discriminant can be enumerated by the following formula:

$$D = a_1^2 - 4 \cdot a_0 a_2$$

After the value of the discriminant is known, the real of the equation roots may be computed:

$$x_1, x_2 = -b \pm \frac{\sqrt{D}}{2 \cdot a_2} \tag{4.4}$$

As we can see, the sign of the discriminant determines the number and multiplicity of the roots. If $D < 0$ then both of the roots are complex and the equation thus does not have any real roots. If $D = 0$ then the roots are equal (thus forming one double root). For values of $D > 0$, the roots are simple - two distinct real numbers.

Cubic Equation A general cubic equation is of the following form (written as a monic polynomial):

$$z^3 + a_2z^2 + a_1z + a_0 = 0$$

According to the fundamental theorem of algebra, the number of real roots of this polynomial may either be one or three (with the possibility that some of them are equal, thus gaining two distinct roots - one of multiplicity two and the other one of multiplicity one - or just one root of multiplicity three).

Analytical solution of this kind of equation is more complicated and, moreover, some of the temporary subresults have to be stored as complex numbers. These facts, combined with the limited computer precision, may cause that the analytically obtained roots are relatively very far away from the real ones. They may even contain (or lack) an imaginary part, even if the actual roots are real (or complex) numbers.

The cubic equation in the above defined form is most often analytically solved by using the so called Cardano's formula, which may be found in [27]. Some simpler solution methods, such as the Vietta's formula, only usable for solving appropriate special cases of the equation are presented as well there.

Quartic Equation Solving the quartic equation, defined as

$$z^4 + a_3z^3 + a_2z^2 + a_1z + a_0 = 0$$

is even more complicated than the case of cubic equation. One can use the Vietta's formulas, described for instance in [30], but the same restrictions as in the previous case apply. The subresults again need to be stored as complex numbers (even if the roots themselves are real), which may theoretically lead to wrong final results.

The number of real roots may be zero, two or four (as defined by the fundamental theorem of algebra, see above). Some or all of them may, again, be equal, thus forming roots of multiplicities greater than one.

4.4 Numerical Methods for Solving Polynomials

4.4.1 General Methods for Solving Nonlinear Equations

With polynomial being a special case of nonlinear equation of one variable, it can be solved by using any suitable method (or combination of several methods) usable for solving these equations. Such methods are often divided into two groups. The first of them contains the methods, which converge slowly, but will converge to a root (if one exists) for any input data (i.e., any suitable function and any initial estimation of the root value). The other group consists of methods that are usually used to increase the precision of a previous root estimation. These methods converge faster in general, but the initial estimation needs to be sufficiently near the actual root, otherwise the method may fail to converge at all.

Another possible way of dividing the numerical methods into groups is the number of initial estimations that need to be passed as input arguments to the method. Some methods need two points that define an interval, which contains the root to be found, other methods need just one initial estimation of the value of the root. Other possibilities exist and may be found in the appropriate literature.

Bisection Is one of the simplest numerical methods, which is often used in various fields of computer engineering (with some modifications). Given an interval $\langle x_1, x_2 \rangle$ and a function $f = f(x)$ such that f is continuous at $\langle x_1, x_2 \rangle$ and

$$f(x_1) \cdot f(x_2) < 0 \tag{4.5}$$

then, due to the law of the mean, at least one root of f lies in $\langle x_1, x_2 \rangle$.

The method of bisection will converge to the root by splitting the given interval into two halves and repeating the process on the half that fulfills the condition 4.5.

Bisection converges for any initial interval that fulfills the above defined conditions, although the convergence is rather slow. In general, the root estimation is improved by one decimal position after three iterations of the method (see [20]).

Regula Falsi This method is very similar to the previous method of bisection. The input values need to fulfill the same conditions, but the interval that contains the root is divided in a different way. Instead of splitting it in half, a line segment is constructed between points $[x_1, f(x_1)]$ and $[x_2, f(x_2)]$ and its point of intersection with the x-axis defines the division of $\langle x_1, x_2 \rangle$. The progress of Regula falsi is illustrated in Figure 4.1.

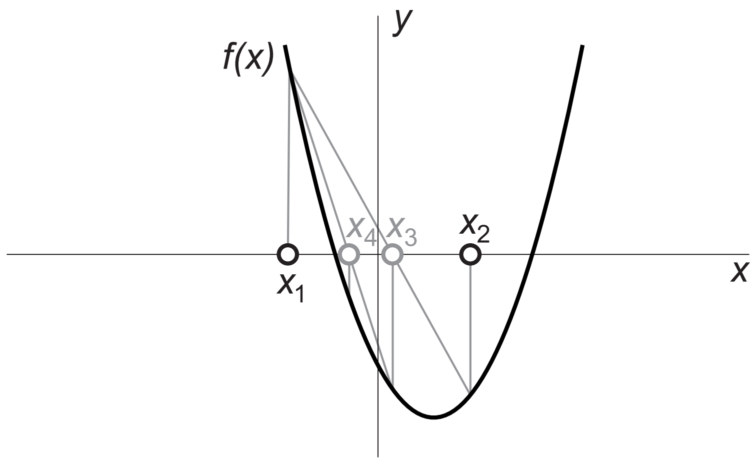


Figure 4.1: The first few iteration of Regula falsi method.

This figure also displays the most significant disadvantage of Regula falsi method - the fact that for convex functions (or convex on the current interval at least), only one of the border points is affected by this method, the other one remains unchanged, thus slowing down the convergence process. Even though this method seems more sophisticated than the Bisection method, it has been shown that their convergence speed is essentially the same (see [20]).

Newton-Rhapson Method Sometimes also called simply Newton's method, this numerical method uses the tangents of the computed function at the current root estimation to improve its precision. The progress of this method is illustrated in Figure 4.2.

Note that Newton's method only needs one root estimation as an input parameter (not an interval like the previous two methods), the formula to compute the next iteration is the following:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{4.6}$$

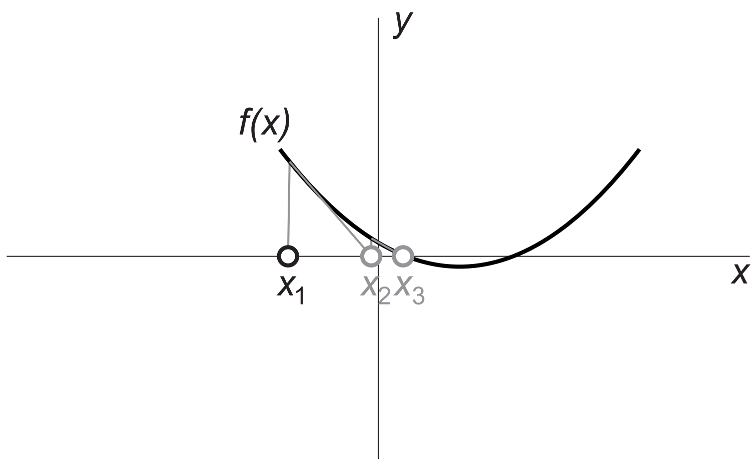


Figure 4.2: The first few iteration of Newton-Rhapson method.

This method converges very quickly, especially when compared to the previous methods, but may not be used on functions that are not differentiable over their domain. And even if this criterion is fulfilled, the method may fail to converge for a bad initial value of the root estimation.

4.4.2 Specialized Methods for Solving Polynomials

Apart from the methods for solving general nonlinear equations, a special sort of methods exist, which are designed to find the roots of polynomials. These methods vary by the type of the roots which they are able to find (real or complex) and the approach they use to do so. Some examples of such functions include (but are not limited to) Lehmer-Schur method, Bairstow's method, Bernoulli's method and others. Details of these methods may be found in literature ([17, 20]).

4.5 Sturm Sequences

4.5.1 Definition

As defined in [20], 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:

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

4.5.2 Construction

Sturm sequence of a polynomial $f(x)$ may be constructed (as proved in [20]):

$$\begin{aligned} f_1(x) &= f(x) \\ f_2(x) &= f'(x) \\ f_{j-1}(x) &= q_{j-1}(x)f_j(x) - f_{j+1}(x), j = 2, ..., m-1 \\ f_{m-1}(x) &= q_{m-1}(x)f_m(x) \end{aligned} \tag{4.7}$$

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. 4.7 to the following form:

$$\frac{f_{j-1}(x)}{f_j(x)} = q_{j-1}(x) + (-1) \cdot \frac{f_{j+1}(x)}{f_j(x)} \tag{4.8}$$

What we see in Eq. 4.8 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 unused for the creation of the Sturm sequence) 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.

4.5.3 Important Features

Counting the Roots Let us define a function $V(x)$ as the count of the number sign changes in the Sturm sequence 4.7 (ignoring all zeros). This function may then be used to count the number of distinct real roots of $f(x)$ on any interval $\langle a, b\rangle$:

$$r_{\langle a,b\rangle} = V(a) - V(b) \tag{4.9}$$

where $a, b \in \mathbb{R}$ or either of a, b may be infinite. As proved in [20], Eq. 4.9 remains valid even if a or b are the roots of $f(x)$.

Root Multiplicity The last term of the Sturm sequence 4.7 may be used to distinguish and compute the values of the multiple roots of $f(x)$. As proved in [20], 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. Note that, if the initial polynomial has some multiple roots, the created sequence is no further a Sturm sequence as defined in Chapter 4.5.1, 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.

5 The Proposed Method

5.1 Overview

The method we propose for handling the Delaunay triangulation of kinetic data is composed of the parts described in this thesis. Several variants of algorithms were considered for each of the subproblem and the best of them were selected, considering not only the "local" point of view, but also the ability to cooperate with the rest of the application without any significant modifications.

Our goal was to implement fully dynamic (with the ability to insert and remove points at any time during the program lifecycle) and fully kinetic (with the ability to remain Delaunay even if the points are moving) Delaunay triangulation of moving points.

5.2 Triangulation Structure and Algorithms

We have considered all the mentioned alternatives for the triangulation data structure, their advantages and drawbacks, with respect to the intended application, and decided to use the incremental insertion algorithm with a walk-based triangle search. This algorithm has several very important features. At first, it is very simple to implement, it is online (we may insert or remove the points during the runtime, thus gaining the ability to simulate discrete time movement) 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 triangulations, we would have to perform relatively large-scale alternations in its structure, which would be prone to errors.

As mentioned above, we used a walk-based triangle search. We have considered several types of walking algorithms and finally we decided to use the Orthogonal walk algorithm in combination with the Remembering stochastic walk (see [14, 22]) as the ideal approach, suggested by [23]. The algorithms of these walks (introduced in the aforementioned resources) follow in Alg. 5.1 for the Orthogonal walk and in Alg. 5.2 for the Remembering stochastic walk. In the case of the Orthogonal walk, only walking in the direction of the x-axis is covered by the algorithm, the following walk along y-axis is very similar.

The main advantage of the Orthogonal walk is the fact that it is really simple and very numerically stable because it does not use any matrix tests to determine which triangle will be next one to search. The remembering stochastic walk then uses a special kind of speedup techniques to increase its performance as much as possible without the lost of the ability to stop the walk in the target triangle. Some walk algorithms use even more speed-up by not testing if they have already reached the target triangle. These walks are stopped after a certain number of steps, which is precomputed based on the size of the triangle mesh and previous experience with such walks. Some other type of walk must then obviously follow in order to pinpoint the target triangle.

Algorithm 5.1: Orthogonal walk algorithm

Input:

- $DT(P)$ - Delaunay triangulation of set of points P
- P_t - target point

Output:

- T_t - triangle containing the target point P_t

Auxiliary:

- T_{curr} - the current triangle of the walk
- T_{prev} - the previous triangle of the walk
- E - an edge of T_{curr}
- P_c - a control point
- P_{start} - starting point of the search

```
// Initialization step
$t_{curr} \leftarrow$ any triangle of $DT(P)$
$P_c \leftarrow$ the point of $t_{curr}$ which is nearest to $P_t$ in $x$-direction
$E \leftarrow$ the edge of $t_{curr}$ which is opposite to $P_c$
$P_{start} \leftarrow$ the midpoint of $E$
$T_{prev} \leftarrow$ the neighbor of $t_{curr}$ over $E$

// Search in the $x$-direction.
if $P_{start}.x \lesseqgtr P_t.x$ then
    while $P_c.x \lesseqgtr P_t.x$ do
        $E \leftarrow$ the edge between $P_{curr}$ and $P_{prev}$
        $P_c \leftarrow$ the point of $t_{curr}$ which is opposite to $E$
        $P_{prev} \leftarrow P_{curr}$
        if $P_c.y \gtreqless P_{start}.y$ then
            $T_{curr} \leftarrow$ the neighbor over the edge to the right from $P_c$
        else
            $T_{curr} \leftarrow$ the neighbor over the edge to the left from $P_c$
        end
    end
end

Proceed similarly for the $y$-direction
Continue with the Remembering Stochastic walk
```

Algorithm 5.2: Remembering stochastic walk algorithm

Input:

- $DT(P)$ - Delaunay triangulation of set of points P
- P_t - target point

Output:

- T_t - triangle containing the target point P_t

Auxiliary:

- T_{curr} - the current triangle of the walk, T_{prev} - the previous triangle of the walk
- E - an edge of T_{curr}, T_{neigh} - the neighbor of T_{curr} over E
- P_{opp} - point of t_{curr}, opposite to E
- $found$ - boolean variable, used as an ending flag of the algorithm

```
$t_{curr} = t_{prev} \leftarrow$ any triangle of $DT(P)$
$found = false$
while $not\ found$ do
    $e \leftarrow$ random edge of $T_{curr}$
    $P_{opp} \leftarrow$ the point of $T_{curr}$ opposite to $E$, $T_{neigh} \leftarrow$ the neighbor of $T_{curr}$ over $E$
    if $T_{neigh} \neq T_{prev}$ then
        // $P_t$ is on the other side of $E$ than $P_{opp}$
        $T_{prev} \leftarrow T_{curr}$, $T_{curr} \leftarrow T_{neigh}$
    else
        $E \leftarrow$ the next edge of $T_{curr}$
        $P_{opp} \leftarrow$ the point of $T_{curr}$ opposite to $E$, $T_{neigh} \leftarrow$ the neighbor of $T_{curr}$ over $E$
        if $T_{neigh} \neq T_{prev}$ then
            // $P_t$ is on the other side of $E$ than $P_{opp}$
            $T_{prev} \leftarrow T_{curr}$, $T_{curr} \leftarrow T_{neigh}$
        else
            $E \leftarrow$ the next edge of $T_{curr}$
            $P_{opp} \leftarrow$ the point of $T_{curr}$ opposite to $E$, $T_{neigh} \leftarrow$ the neighbor of $T_{curr}$
            over $E$
            if $T_{neigh} \neq T_{prev}$ then
                // $P_t$ is on the other side of $E$ than $P_{opp}$
                $T_{prev} \leftarrow T_{curr}$, $T_{curr} \leftarrow T_{neigh}$
            else $found = true$
        end
    end
end
// Now $T_{curr}$ contains $P_t$
```

5.3 Movement Approach

Even though the discrete time approach, described in the previous text, is relatively simple to implement (and is, in fact, supported by the online feature of the triangulation and the ability to remove the points from it), we did not use it as the main means of point movement. We did so, because of its mentioned disadvantages, which represent the possibility of missing some vital events in the lifecycle of the triangulation.

We rather chose the continuous movement approach with the continuous legalization of the triangulation structure as described in Chapter 3.3.4. The other continuous movement approaches also represent a possibility, but they also possess a potential source of errors with the fact that the triangulation they manage loses and regains the Delaunay property, which may be a kind of unwanted behavior.

As mentioned before, in the case of this movement approach, it is necessary to compute the times of topological events by solving the time dependent incircle test matrix (see Eq. 3.7). The determinant of this equation is in the form of a polynomial - see Eq. 2.1.

5.4 Polynomial Solving

5.4.1 Available Methods

From the polynomial solving methods, analytical methods may be dismissed, because they do not allow the use of any inbuilt precision enhancement methods. For higher degrees of polynomials (i.e., the third and the fourth degree in our case), the subresults are complex numbers, which not only reduce the precision, but also may cause the shift of real roots away from the real axis in the complex plane and vice versa. For the first and the second order polynomials, it is possible to use the analytical solution because of its extreme simplicity. The described numerical methods specialized on polynomial solving are also unsuitable for our purpose because of very similar reasons as in the case of analytical methods. These methods are either designed to search for the roots in the complex plane (which may again cause real to complex shifts) or are unnecessarily complicated, oriented to solve generic polynomials (with no upper bound on the degree of the polynomial) and thus contain computation substeps, that are time- and precision-consuming and present no useful information on the polynomial in the computation process.

5.4.2 The Proposed Method Concept

Instead of using one of the described methods, which did not suit our purpose well enough, we introduced our own method for polynomial solving, based on the idea proposed by [5]. This method is based on the information that can be obtained about a polynomial from its Sturm sequence and will be described further. Even though the Sturm sequence may provide us with quite a valuable information about the polynomial, it does not suffice to solve the polynomial by itself. It has to be combined with any other suitable numerical method or methods. In this case, we propose the 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 the Newton's method.

5.5 Using Sturm Sequences to Solve Polynomials

5.5.1 Initial Conditions

Let us assume that a polynomial has been passed as an argument to the function that should compute its roots. If the order of the polynomial is lower than three, we compute its roots analytically. If the polynomial order is equal to three or four, we process to the creation of its Sturm sequence, determine the count and multiplicities of its roots and then solve it using either numeric or analytical methods, based on its recognized features.

Furthermore, because we want to use the polynomial to determine the times of future topological events, we are only interested in the roots that are greater than or equal to zero [1] (or some current time value of the triangulation).

5.5.2 Polynomial Solving

Together with the fundamental theorem of algebra, we may use the knowledge obtained from the Sturm sequence of a polynomial to create a table of guidelines for its solving. As said before, the last polynomial of each sequence may be used to discover all the multiple roots of the solved polynomial. The guidelines are presented in Table 5.1:

$\deg f(x)$	$f_m(x)$ real root mult.	$f(x)$ real root mult.
3	{2}	{3}
3	{1}	{2, 1}
3	none	{1} or {1, 1, 1}
4	{3}	{4}
4	{2}	{3, 1}
4	{1, 1}	{2, 2}
4	{1}	{2} or {2, 1, 1}
4	none	{1, 1} or {1, 1, 1, 1}

Table 5.1: Features of the polynomial depending on its Sturm sequence

In this table, the first column determines the degree of the solved polynomial $f(x)$, the second column shows the multiplicities of the roots of the last polynomial in the Sturm sequence constructed for the solved polynomial. The last column then shows all the possible root multiplicity configurations for the solved polynomial. For instance, if the polynomial $f(x)$ is of the third degree and the last polynomial in its Sturm sequence has one simple root, then $f(x)$ has to have one double root and no other roots of multiplicity greater than one. Furthermore, according to the fundamental theorem of algebra, number of complex roots of a polynomial must be either even or zero. It may not be even, because only one root is left to recognize, thus this remaining root must be real, leaving $f(x)$ with only one

[1] In general, we may not ignore roots that are equal to zero, because the execution of some topological events may cause scheduling of another event that takes place at the exact same time instant. A typical example of this is the configuration of five cocircular points.

possible root configuration - one double real root and one single real root, as shown in the second row of Table 5.1. Examples of all possible root configuration of a polynomial of the third degree are shown in Figure 5.1. This figure shows examples of all the root configurations of a polynomial, as presented in the first three rows of Table 5.1, without the corresponding Sturm sequences.

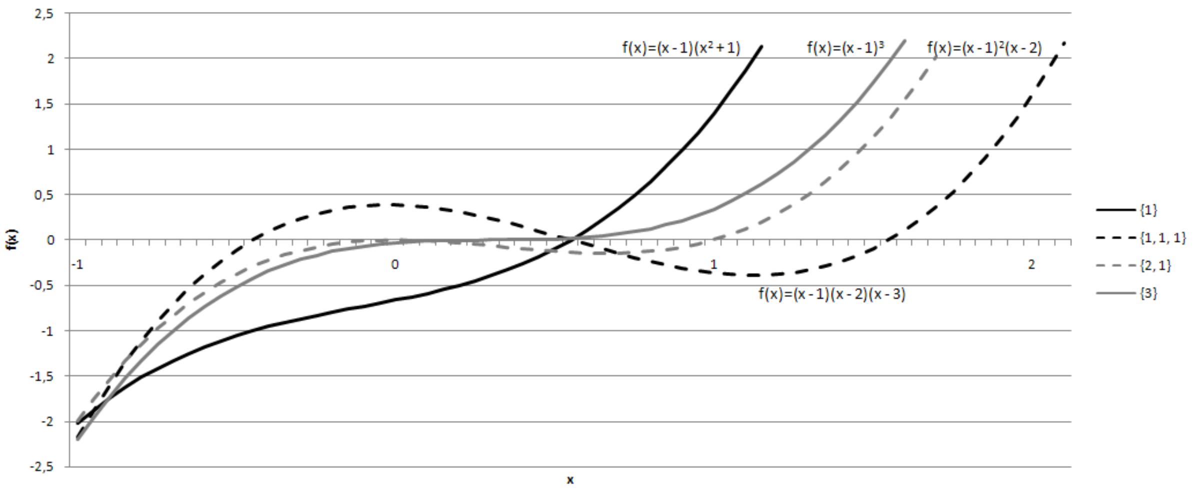


Figure 5.1: Examples of all possible root configurations of a polynomial of the third degree.

The whole process of the polynomial solving may then be summarized into the *Sturm*3 algorithm (see Algorithm 5.3), also described in [25]. This algorithm shows how to solve a polynomial of the third degree by using the proposed method. The idea for higher degree polynomials remains the same, but the number of possible root configurations grows larger as shown in Table 5.1. In this algorithm, we can see that the situation is quite simple for the polynomials with any number of multiple roots. If we recognize that the $p(t)$ has one triple root, there are no other roots left to compute and no numerical method is necessary. If $p(t)$ has one double root, we use the polynomial root decomposition as described in Chapter 4.2.3 and solve the remaining linear equation analytically. The situation is a little more complicated for polynomials without any multiple roots - we have to use the precomputed value of r (which denotes the total number of the roots of $p(t)$) and then solve $p'(t)$, which may be done either analytically (the derivate of a polynomial of the third degree is a quadratic equation) or for instance recursively (for higher degree polynomials) by calling the lower order versions of this algorithm. The roots of the derivate then determine the intervals, which contain the roots of $p(t)$. Note that the derivate $p'(t)$ of $p(t)$ may have two roots, thus defining three intervals, even if $p(t)$ has only one single root. In this case we have to check the signs of the values of $p(t)$ for each of the roots of the derivate polynomial before running the numerical methods. The outer bounds of the positions of the roots may be obtained for instance by the method shown in Chapter 4.2.1.

Algorithm 5.3: Sturm3 Algorithm

Input:

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

Output:

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

Auxiliary:

- Sturm sequence $f_1(t), ..., f_m(t)$ of the polynomial $p(t)$ - see Eqs. (4.7), note that $f_1(t) = p(t)$.
- $R_m = \{r_{mi}\}_{i=1}^{r_{mult}}$ - a sequence of all the multiple roots of $p(t)$. Each multiple root r_{mi} is contained $m_i - 1$ times, where m_i is its multiplicity.

```
// Create the Sturm sequence
$f_1(t), ..., f_m(t) \leftarrow$ Sturm sequence of $p(t) = f_1(t)$
$r \leftarrow (V(-\infty) - V(\infty))$// See Eqn. 4.9
if $r = 0$ then
    // $p(t)$ has no real roots
    // This situation may not occur for the polynomials of the third
       degree, but is possible for the polynomials of even degree.
    Return empty sequence $\{\}$ of roots.
end
// Obtain the multiple roots of $p(t)$
$R_m \leftarrow$ sequence 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
    // $p(t)$ has a double and a single root (see Tab. 5.1)
    $r_s \leftarrow$ the only single root of $\frac{p(t)}{(t-r_{m1})^2} = 0$
    Return $\{r_{m1}, r_{m1}, r_s\}$// A double and a single root
else
    // No multiple roots, solve $p(t)$, using a suitable numerical method
    Return $\{r_i\}_{i=1}^{r}$ ... sequence of $r \in \{1, 3\}$ distinctive roots.
end
```

5.5.3 Special Point Configurations

Sometimes it is useful (and in some cases it is even necessary) to know some special point configurations, because they may relatively significantly influence the computations in progress. As told before, the complexity of the polynomial (i.e., its degree and the count and multiplicities of its roots) is strongly affected by the mutual position and velocity vectors of the four points which define the polynomial in question. It is thus logical that certain configuration lead to degenerate cases of the polynomials.

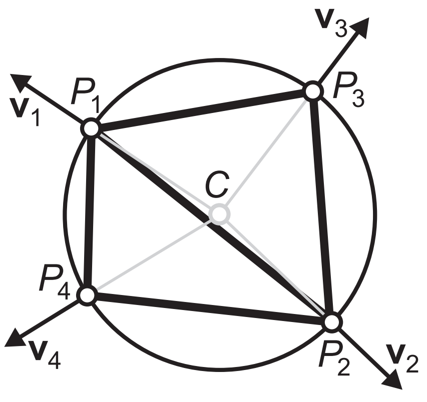


Figure 5.2: Four cocircular points moving away from the center of their circumcircle.

Figure 5.2 shows the first sort of special point configurations. If the displayed points P_1, P_2, P_3 and P_4 are cocircular and moving with velocity vectors $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}$ and $\mathbf{v_4}$ as shown in the figure - with C being the center of their circumcircle and the only point of intersection of all their four trajectories, then the following two cases may occur (also described in [25]):

1. $\mathbf{v}_1 = \mathbf{v}_2 = \mathbf{v}_3 = \mathbf{v}_4 = \mathbf{0}$

   In this case, all the points are cocircular and not moving. Eq. 2.1 then degenerates into $0 = 0$ and cannot be solved. No edge swaps are required, because the points are static and both available triangular configurations for four cocircular points are legal for the planar Delaunay triangulation.

2. $\|\mathbf{v}_1\| = \|\mathbf{v}_2\| = \|\mathbf{v}_3\| = \|\mathbf{v}_4\| \neq 0$

   The points move away from their circumcenter equally fast. This means that there will be a topological event for each $t \in \mathbb{R}$ as the circumcircle will grow. We may discard all of the obtained topological events because both possible triangle configurations are legal due to all four points lying on the same circle and thus no edge swapping is necessary. A similar situation arises when the points are all moving towards their circumcenter.

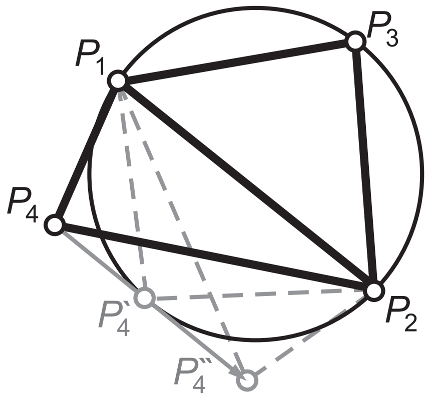


Figure 5.3: Tangential movement of P_4.

Another, and even more important, special point configuration is shown in Figure 5.3. Point P_4 moves tangentially to the circumcircle defined by triangle $P_1P_2P_3$. As we can see, only one of the two triangle configurations is legal for each time, except for the one time instant, where the four points become cocircular. At this single instant, both the available configurations are legal. According to these facts, no edge swap is necessary and these events may be ignored. In the case of this one and similar point configurations, the solved polynomial will have roots of multiplicity greater than one, that determine their times. We may then state the following hypothesis:

Hypothesis 5.5.1. *The real roots of polynomial (2.1) may be divided into two groups as follows:*

i All roots of even multiplicity may be ignored when determining the times of topological events.

ii All roots of odd multiplicity determine the time of a single topological event.

The simplest case, when all the roots are of multiplicity equal to one is self evident - in this case, each of the roots specifies the time when the determinant of $\mathbf{I} = \mathbf{I}(t)$ is equal to zero and thus the four points lies on a circle. Now, let us consider the following case:

- Let $p(t) = 0$ be a polynomial as defined in Eq. 4.1
- Let $t_1, t_2 \in \mathbb{R}$ be two distinct real roots of $p(t)$ of multiplicity one such that no other root t_3 lies between them:

$$\exists t_1, t_2 \in \mathbb{R}, t_1 < t_2 : p(t_1) = 0 \wedge p(t_2) = 0 \Rightarrow \forall t_3 \in (t_1, t_2) : p(t_3) \neq 0$$

Each of these roots denotes an edge swap in the triangle pair as a result of triggering a topological event (see Chapter 3.4.4). Only two triangular configurations are possible and to process a topological event means to switch between them. Let us mark τ_1 the configuration that is legal during time intervals $(t_1 - \varepsilon_1, t_1\rangle$ and $\langle t_2, t_2 + \varepsilon_2)$[2] and τ_2 the configuration legal during $\langle t_1, t_2\rangle$. Note that both τ_1 and τ_2 are legal for $t \in \{t_1, t_2\}$

Now let us alter the positions $P_1, ..., P_4$ and velocity vectors $\mathbf{v_1}, ..., \mathbf{v_4}$ of all the points by adding $\Delta P_1, ..., \Delta P_4$ and $\Delta\mathbf{v_1}, ..., \Delta\mathbf{v_4}$, respectively, to their values in such a way that only the position of root t_1 is altered by Δt_1 (let $t_1 < t_1 + \Delta t_1 < t_2$) and the positions of all other roots $t_2, ..., t_n$ of polynomial $p(t) = 0$ remain unchanged (let us assume without a proof that such a new values of the point coordinates and the velocity vectors do exist).

[2]Let $\varepsilon_1, \varepsilon_2$ be such small that these intervals contain no other roots of $p(t) = 0$

Let us mark the original point position and velocity configuration by vector ξ_0 and the changes of this configuration by $\Delta\xi_0$. Let $\{\xi_i\}_{i=0}^{\infty}$ be a sequence of such vectors that $\xi_{i+1} = \xi_i + \Delta\xi_i$ and that, if t_1^i is the value of the root t_1 for the configuration ξ_i, then $\forall i : t_1^{i+1} > t_1^i$. There is a limit $\hat{\xi}$ to this sequence which represents such a vector of point positions and their corresponding velocity vectors, where $t_1 = t_2$ is one double root of $p(t) = 0$.

$$\lim_{i\to\infty}\{\xi_i\} = \hat{\xi}$$
$$\lim_{i\to\infty}\|t_1^i - t_2^i\| = 0$$

As we can see, the length of the interval $\langle t_1, t_2\rangle$ where τ_2 is a legal triangle configuration converges to zero. This means that in the limit case the τ_2 is legal only for time $t_1 = t_2$ which is the double root of $p(t) = 0$. According to this fact, a single edge swap is an option only if the triangulation is about to remain in the singular case, but it is not necessary because the other triangle configuration τ_1 is legal during the whole interval $(t_1 - \varepsilon_1, t_1\rangle \cap \langle t_2, t_2 + \varepsilon_2) = (t_1 - \varepsilon_1, t_2 + \varepsilon_2)$ as defined before.

The approach is essentially very similar for the roots of multiplicities greater than two.

5.6 Numerical Stability

5.6.1 Singularities in the Triangulation

As described earlier in Chapter 3.4.4, the movement of points may lead to an occurrence of a singular case in the triangulation. Besides the tangential movement, there is a possibility of two point collisions. As previously described in [25], this situation leads to the two points being cocircular with any other two points in the triangulation, which is most unwanted because of the way of obtaining the topological events. Large number of nonexistent events may be scheduled at the time of such singularities and they may then destroy the triangulation structure.

5.6.2 Ordering the Events in Queue

If the points represent some physical objects and thus are subject to some forces as a result of their mutual affection, it is possible to order the events in the queue in such a way that if two events of different kinds (e.g., a collision event and a cocircular event) should occur at the same time, the collision event will be processed first. This precaution will cause that one of the two colliding points will be deflected from the other by the reactive forces, which may help to stabilize the singular situation. The remaining topological events may then be removed from the queue and replaced by new events, which take into account the new velocity vectors of the two colliding points.

5.6.3 Safety Discs

Safety discs may be added to each of the points in the triangulation to prevent any two of them to become too close. These discs represent an area around a point, which may not be entered by any of the other points. A minimal distance between points is then introduced, which is equal to the diameter of the discs (if all the discs have the same diameter). This minimal distance between points ensures that the singularities will not occur and may be very well combined with the previous method of ordering the events in the queue.

5.6.4 Randomization

It is sometimes impossible to introduce any of the aforementioned methods, but the singular cases must be dealt with. In some of these cases the randomization may be used. By altering the positions or velocity vectors (or even both of them) by adding a small random value, very good results may be obtained. The random factor helps to separate the events which would otherwise take place at the same time and thus prevents the singularities. Due to the limited precision of the floating point numbers in computer arithmetics, the randomization of the position of points is likely to prevent vast majority of collisions between points.

5.7 Other Types of Movement

5.7.1 Overview

Even though the restrictions we made on the type of point movement may seem very serious, the available resources may still be sufficient for more complex types of movement. The following chapter provides information about two such methods - the first of them binds several points to point clusters, which share a common velocity of the points and thus their mutual positions are stable and the second one provides a simple way to simulate nonlinear movement only by using the tools and techniques described before. Finally, a movement along polynomial trajectories is considered and some propositions are made on the construction of the required mathematical apparatus.

5.7.2 Point Clusters

From the practical point of view, the point clusters may represent a simplification of some complex objects, or some unification of points that share some common feature. Figure 5.4 shows a simple example of clustered data.

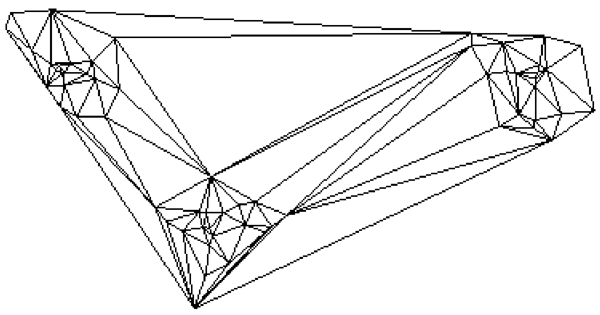


Figure 5.4: A simple example of triangulation of a set of clustered data.

Let us define a point cluster as a set of n points $C = \{P_1^C, ..., P_n^C\}$ that move along colinear trajectories with identical velocity vectors $\mathbf{v_1^C} = ... = \mathbf{v_n^C}$. Then the described methods for handling Delaunay triangulation of kinetic data suffice fully for managing point clusters if some method of simultaneous velocity vector alternation is introduced, meaning that if the velocity vector $\mathbf{v_j^C}$ of a point P_j^C changes to $\mathbf{v_j^C}'$, then the velocity vectors of all the remaining points in the cluster are changed to the exact same values immediately. This may be done for instance by scheduling collision events (or some other type of events, which cause some changes of the velocity vectors) for all the points in a cluster, but upon the execution of the first of them, all of the events are executed in the same fashion. For instance, if the whole cluster is moving in the direction of x axis and the rightmost point in the cluster collides with the right wall of the bounding are, then all the velocity vectors of the clustered points are changed as if these points collided with the wall themselves.

5.7.3 Nonlinear Trajectories through the Linear Interpolation

If given a parametric equation of a nonlinear function $f = f(t)$, we are able to divide the range of the parameter t into small intervals $\langle t_i, t_{i+1} \rangle$ and replace $f(t)$ by line segment on each of these intervals, as shown in Figure 5.5. The size of the intervals may be selected in any suitable way, depending on the used nonlinear function, previous experience or maximum allowed error of the interpolation. The intervals may even vary in size.

In order to force a point P' to move along the nonlinear trajectory, its velocity vector must be modified at each time instant t_i. The velocity vector $\mathbf{v_i}$ for each $t_i, i < n$ will then be computed as follows:

$$\mathbf{v_i} = \frac{1}{\|\langle t_i, t_{i+1} \rangle\|}(P_{i+1} - P_i) \tag{5.1}$$

where $P_i, i \leq n$ are the positions of the moving point at t_i, we can compute their coordinates simply by using the parametrization of $f(t)$:

$$P_i = [f_x(t_i), f_y(t_i)] \tag{5.2}$$

where $f_x(t)$ and $f_y(t)$ are the parameterizations of $f(t)$ for the x and y axes.

The maximum error of such an interpolation may be then bound by the value $E(t)$ as

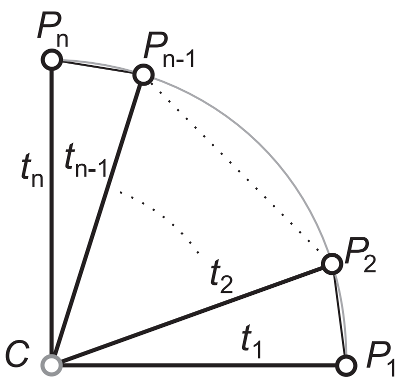


Figure 5.5: Linear interpolation of a circular trajectory.

follows in Eq. 5.3, if certain conditions on the interpolated function are met, see below the equation.

$$|E(t)| \leq \frac{M}{(n+1)!}|e_n(t)| \leq \frac{M}{(n+1)!}\max_{t\in\langle a,b\rangle}|e_n(t)| \tag{5.3}$$

where n is the order of the interpolation polynomial (in the case of a linear interpolation $n = 1$), M is a constant (see below), $e_n(t)$ is the interpolation function on $\langle a, b\rangle$.

As stated above, Eq. 5.3 may only be used under certain conditions, these conditions are:

- $f(t)$ must be continuous on $\langle a, b\rangle$.
- $n + 1$ derivations of $f(t)$ must exist on $\langle a, b\rangle$
- Such constant M must exist that $M \geq |f^{(n+1)}(t)|$ for each $t \in \langle a, b\rangle$

For more details on the linear interpolation method and its error, see [19].

5.7.4 Polynomial and Nonlinear Trajectories

Considering the fact, how the trajectories of the moving points affect the equation, which has to be computed in order to obtain the times of topological events (see Eqs. 3.7 and 2.1), it is obvious, that increasing the order of the polynomials that create the trajectories of the points will not change the nature of these equations.

If the points move along trajectories determined by the polynomials $p_x(t), p_y(t)$ of degree up to n (both in x and y coordinates), the determinant of Eq. 3.7 will be a polynomial of degree $4n$ (considering 2D case only, for higher dimensions, the polynomial degree will grow even more).

However, the described mathematical methods may be generalized, with some effort, for these high degree polynomial cases, but the number of possible root configurations will grow, for instance even the fifth degree polynomial has 11 available root configurations (as in the fashion of Table 5.1).

General nonlinear trajectories without any form of aforementioned simplification represent even more complex problem. They require solving of nonlinear equations without any known parameters (considering the case, where the user is free to insert any nonlinear function as a trajectory). This kind of movement is beyond the reach of the proposed method and must be handled in a completely different way. The shown mathematic relations still hold, but the equation that describes the time dependent determinant of the incircle test matrix (see Eq. 3.7) will be of general nonlinear nature and thus cannot be solved by using the method based on Sturm sequences of polynomials. The simplest way to handle these points may be the discretization of time, thus using the dynamic, rather than kinetic, property of the triangulation, with all the consecutive drawbacks of such an approach.

6 Performance

6.1 Implementation Details

All the mentioned test results were obtained by running our implementation of the kinetic data structure in *C#*, compiled with *Microsoft Visual Studio 2005* on a PC with the following HW/SW configuration:

Operation System: Microsoft Windows XP, SP 2

Framework Version: Microsoft .NET Framework 2.0.50727 SP 1

DirectX Version: Microsoft DirectX 9.0c (March 2008)

CPU: Intel Pentium M Processor 1.73 GHz

RAM: 512 MB

Video Adapter: ATI Mobility Radeon X700, 128 MB RAM

The application, which demonstrates our implementation of Delaunay kinetic triangulation may be found on the enclosed CD. Figure A.1 in Appendix A shows some screenshots of its user interface. The application allows user to insert and remove points into and from a triangulation, lets him assign them the velocity vectors or nonlinear trajectories and allows the binding of points into clusters. When the time of the triangulation is increased, the displayed structure is updated by using the method we proposed in this text.

As the test set, we used 100x100 units bounding area, additional details such as the count and positions of points used during a test are added to each of the test results. Each test was performed 10 times and the presented results represent average values from these experiments.

6.2 Tests and Results for Simple Data Sets

6.2.1 Overall Time Complexity

The simple data tests were performed on randomized data - 100 random points with safety disc of 1 unit radius were inserted into the triangulation, certain percentage of them were assigned a random [1] constant velocity vector with both its coordinates being a random number in interval $\langle -5, 5\rangle$, and each of the configurations was measured for 10 seconds of the inner time of the triangulation. The measured values include the insertion of points into the triangulation, the initialization phase at the start of the movement and then the movement itself.

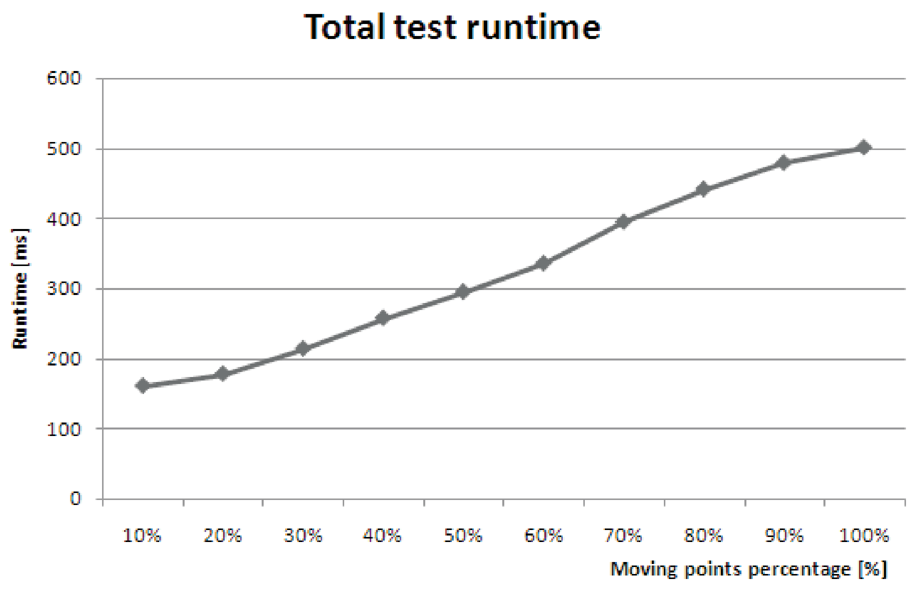


Figure 6.1: Total runtime needed for the test with constant total amount of points.

The graph in Figure 6.1 demonstrates the dependency of the total time required for the test on the percentage of the moving points (the total amount of point remains unchanged). According to the measured values, we may assume, that there is an upper bound of $O(n)$

[1]Each random variable in the tests was generated by using a random number generator with uniform distribution of probability.

and lower bound of $O(\log n)$ on the runtime needed if the total amount of points remains unchanged and n represents only the moving point percentage.

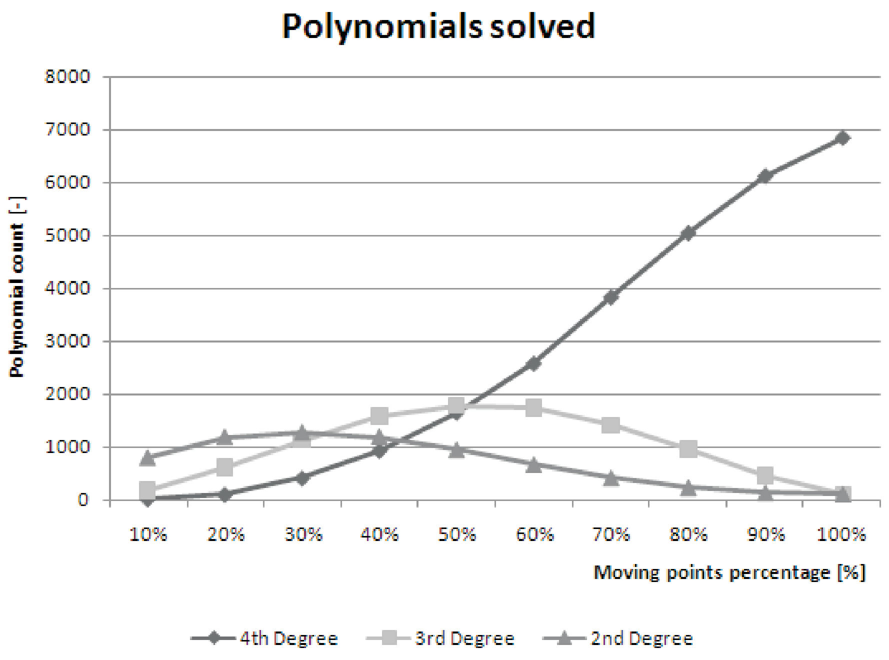


Figure 6.2: Number of polynomials of different degrees computed during the test.

The graph in Figure 6.2 shows the number of polynomials of the degree from two to four that had to be computed during the test. It is obvious that the average degree grows with the growing percentage of moving points. This fact is a consequence of the dependencies shown earlier in Chapter 5.5.3, approving the fact that the velocity vectors of the four points in the two triangle configuration strongly affect the degree of the solved polynomial. The consequences of this behavior are that the performance of the entire kinetic part of the application will be strongly affected by the performance of the method used to compute the relevant polynomials. And because we used a method, which solves the polynomials of the third and the fourth degree by using iterative methods (the bisection and the Newton's method), we may expect significant performance loss for higher percentages of moving points.

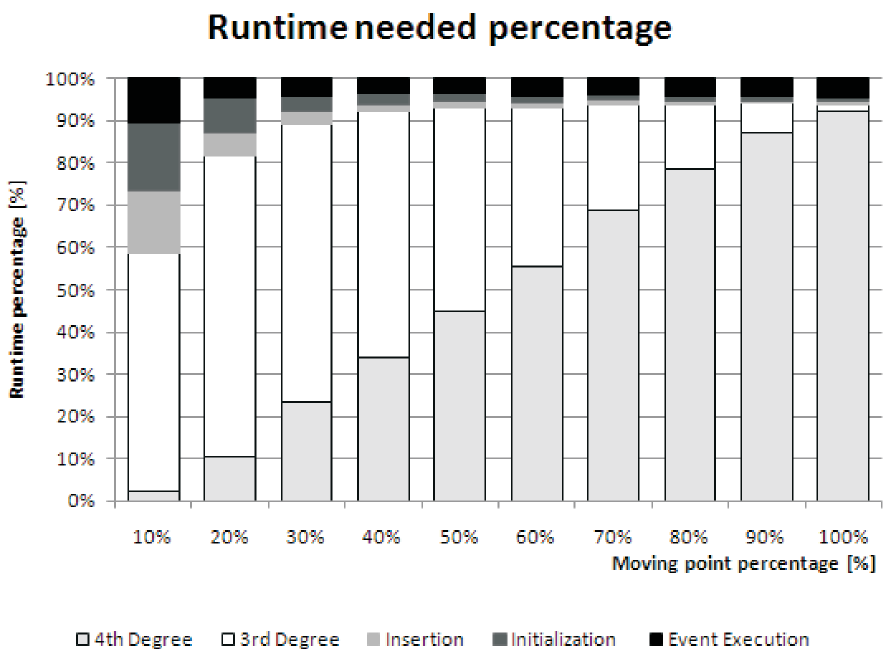


Figure 6.3: Runtime percentage consumed by different parts of the application.

The graph shown in Figure 6.3 displays the percentage of runtime consumed by different parts of the application. The displayed values correspond perfectly to the results shown in the previous graph in Figure 6.2. We may see that for any number of moving points in the triangulation, the iterative methods for computing the polynomials of degree greater than two consume an inconsiderable amount of total runtime percentage. It is thus obvious that any further runtime improvements will have to start by either reducing the total number of counted polynomials or improving the overall performance of these methods.

6.2.2 Event Execution

The graph in Figure 6.4 shows the counts of executed events of different kinds and the number of discarded cocircular events. The measured values show that there is an upper bound of $O(n)$ and a lower bound of $O(\log n)$ to both the number of executed and discarded cocircular events. The other types of events represent only a minor part of the total event count.

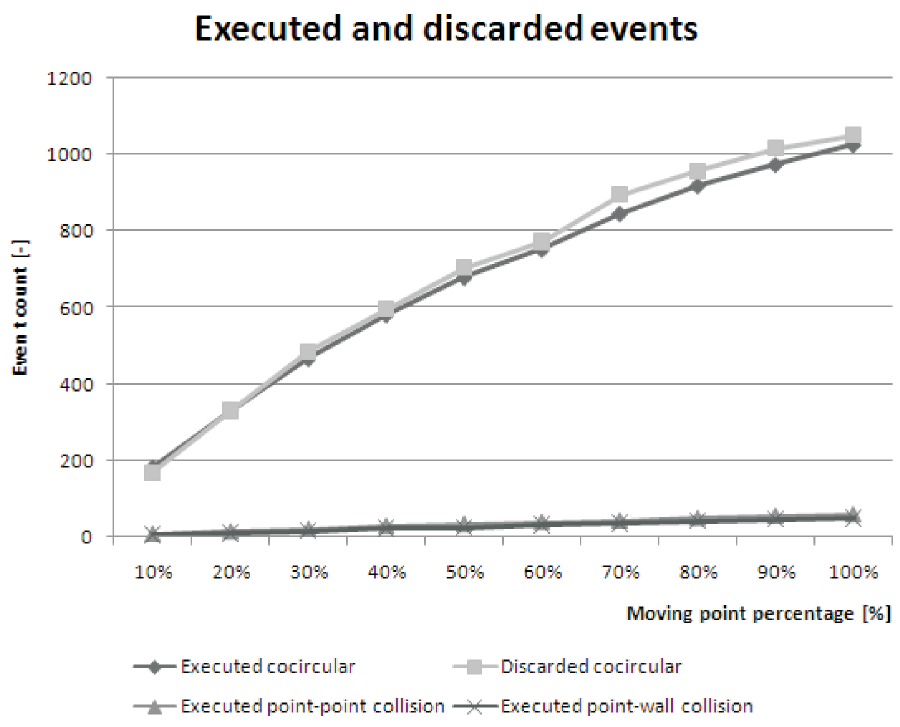


Figure 6.4: Number of executed and discarded events of various types.

Values in this graph show that approximately half of the events that are popped from the queue for execution is discarded because they are no more valid at that time. Because these events have to be computed, but are not used, they represent a kind of a wasted runtime. As stated before, one of the further optimization possibilities lies in the reduction of the amount of polynomials which have to be solved. Byproduct of this optimization technique would also be the reduction of discarded cocircular events.

6.2.3 Queue Performance

Because the whole algorithm, and thus even the encapsulating application, is based on a priority queue, its performance should be observed. The following graphs in Figures 6.5, 6.6 and 6.7 show the amount of events of different times stored in the queue during the whole test for 10, 50 and 100 moving points respectively (the total amount of points in the triangulation is 100 for all the three tests).

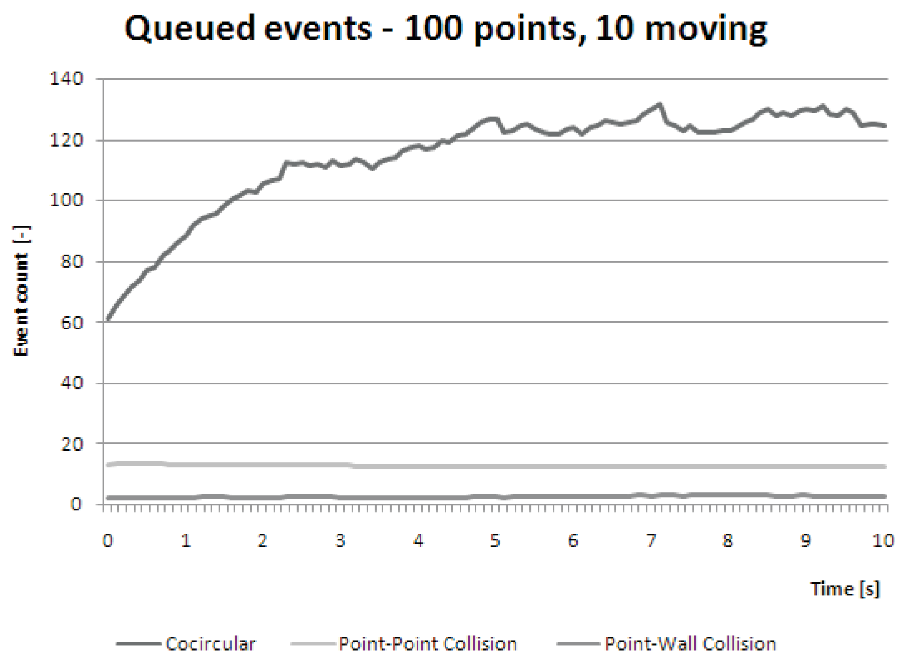


Figure 6.5: Number of events stored in the queue (10 moving points).

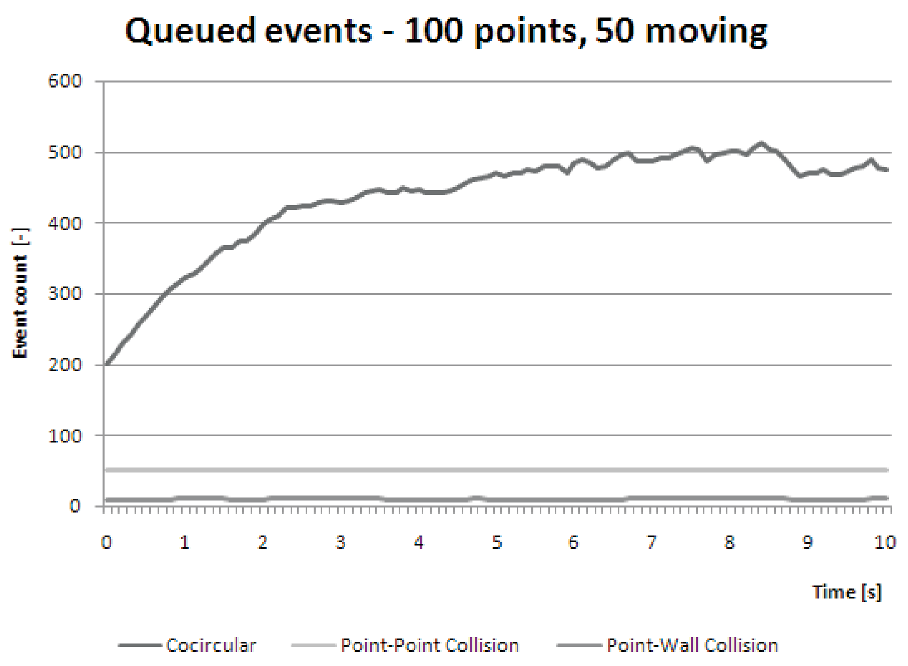


Figure 6.6: Number of events stored in the queue (50 moving points).

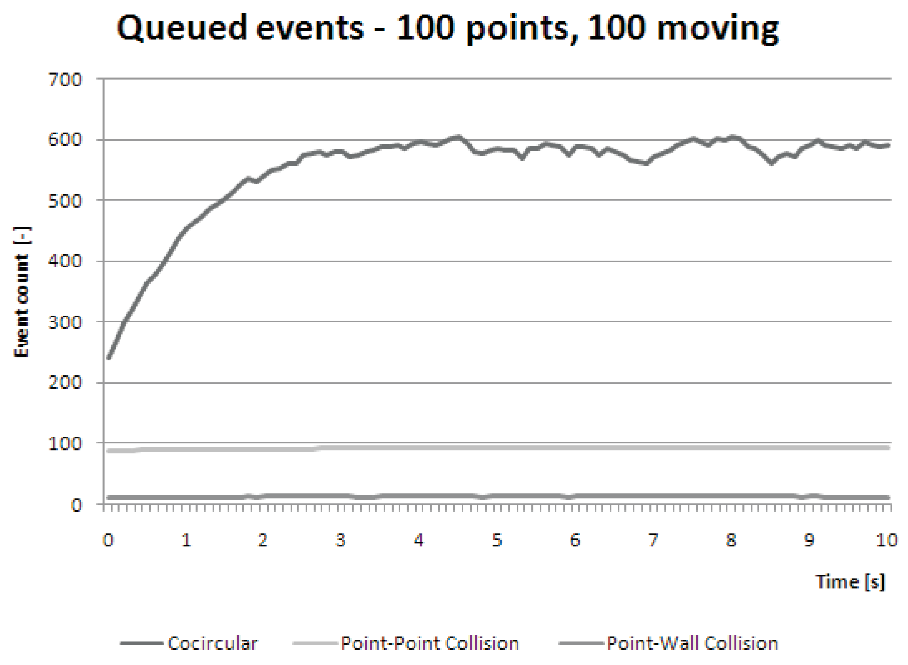


Figure 6.7: Number of events stored in the queue (100 moving points).

Values in the three aforementioned graphs show that there is obviously an upper bound of $O(n)$ and lower bound of $O(\log n)$ on the cocircular events in the queue and the number of the other types of events is approximately constant. We may notice that the count of the queue events grows for some short period of time (in all the displayed cases) and then, after reaching certain value, which is different for each of these graphs, seems to be approximately constant. This behavior is probably caused by the fact, that events are not removed from the queue after they become invalid, so after some time the queue contains number of events, that are relatively far in the future but will never be executed. Based on the measured values, we may assume that these events are distributed in such a fashion that the execution of legal topological events and the resulting scheduling of new topological roughly compensates the count of events discarded from the queue because of being invalid. This theory also explains the discrepancy between the limits of the numbers of the queued topological events in these graphs and the counts of the discarded and executed topological events as shown in the graph in Figure 6.4, because it allows us to determine the number of different topological events that were queued during the time of the test. Let us for instance consider the case for 10 moving points. From the values in Figure 6.4, we can see that approximately 200 topological events were executed and roughly the same amount of this type of events were discarded, additionally, the graph in Figure 6.5 shows that at the end of the test, approximately 120 events were queued. These values let us compute that roughly 520 different events were pushed into the queue during this test.

The values in the graphs in Figures 6.5, 6.6 and 6.7 may not be used to estimate the values in the graph in Figure 6.4 and vice versa, because they are in fact independent to a certain degree (some dependency is present and shown in the following graph). The sole number of queued events at a time instant does not allow us to presume anything about how many of them will be discarded and how many of them will be executed.

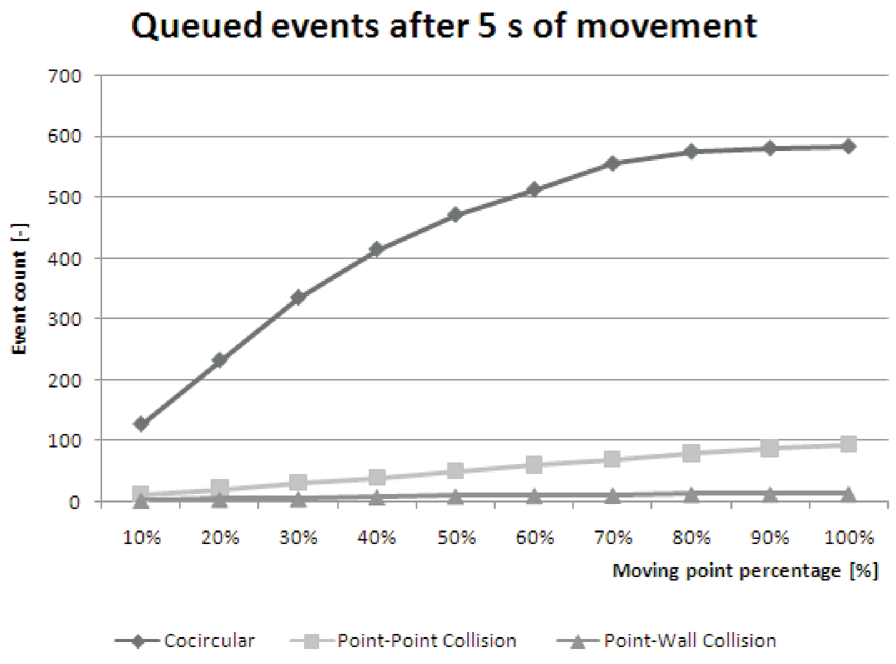


Figure 6.8: Number of events stored in the queue after 5s of the movement.

The graph in Figure 6.8 shows the dependency of the number of the different types of events in the queue on the moving point percentage in the test. The values were obtained by measuring after 5 seconds of the test. The values in the graph show that the approximate limit value of the queued events shown in the previous three graphs (in Figures 6.5, 6.6 and 6.7), depends on the percentage of the moving points in the triangulation with lower bound of $O(\log n)$ and upper bound of $O(n)$.

6.2.4 Realtime Capabilities

The tests in this chapter were performed in order to pinpoint the current realtime capabilities of the application. Their purpose was to determine how many points may be inserted into the triangulation if the test should consume lower amount of runtime than the inner time of the triangulation at the end of the test. The triangulation, HW and SW settings remained unchanged for these tests, but the number of total points inserted and the moving point percentage was changed (the details are provided for each test separately).

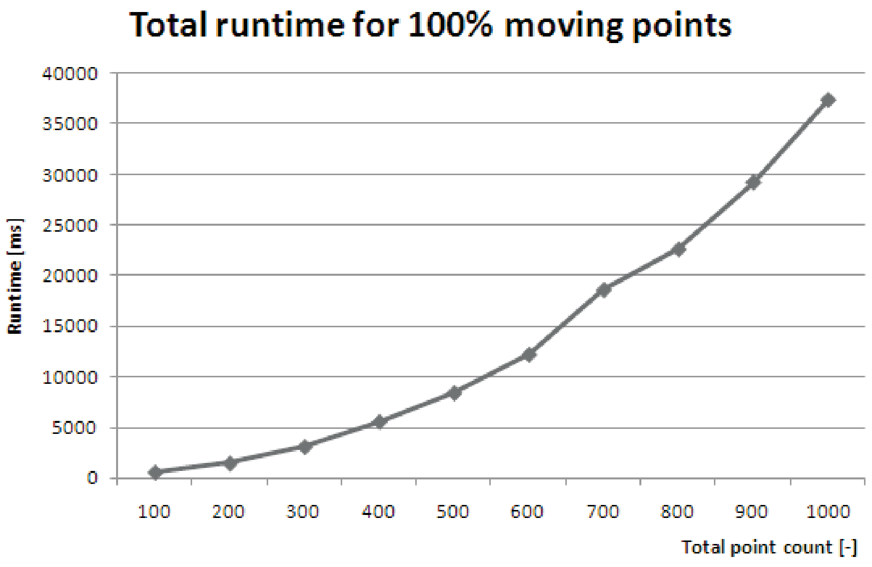


Figure 6.9: Total runtime needed for the test with 100% moving points.

In this test, all the point in the triangulation were moving and the total number of inserted points was being increased from 100 to 1000 as shown in the graph in Figure 6.9. The measured curve is a relatively very good approximation of a parabola, which shows that the time complexity of the runtime needed is of $O(n^2)$ order if n represents the total amount of points in the test and if all of these points are moving (note the difference - unlike the previous tests, n denotes the total number of points in the triangulation). From the measured values, we may state that the current maximum realtime capabilities for the described configuration are somewhere around 500 moving points in the triangulation, but the situation may differ quite significantly for a larger (or smaller) triangulation area or a different scale of the movement vectors or a different radius of safety discs.

6.3 Nonlinear Trajectories

The tests of on points with nonlinear trajectories were performed on the same point sets as the previous cases, but the safety discs around points were not present. The nonstatic

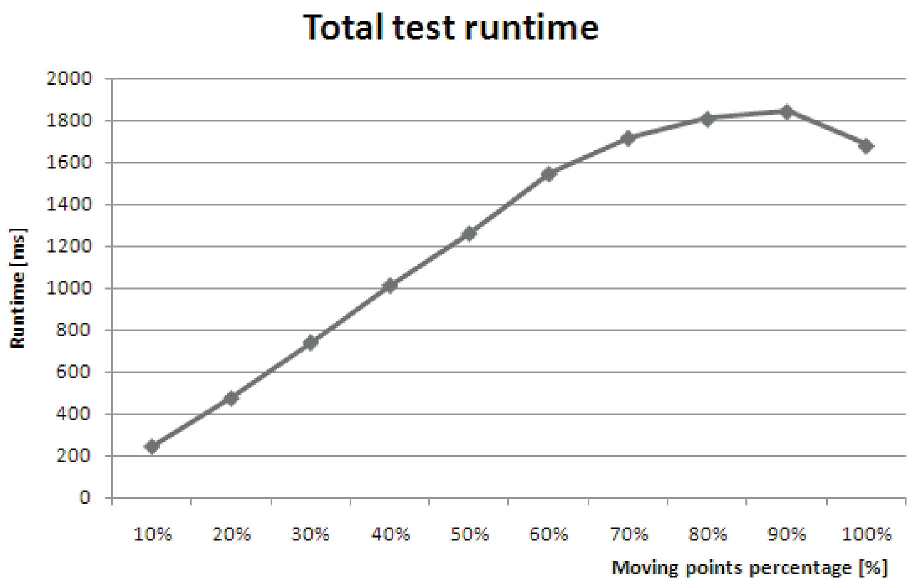


Figure 6.10: Total runtime needed for the test with constant total amount of points with elliptic trajectories.

points were moving along elliptic trajectories (which were obtained in the way described in Chapter 5.7.3) with radii in both axes being the lower of two numbers - 1 or half the distance from the point to the boundary of the triangulation area and with identical phase values. Both these precautions were taken with an intension to minimize the number of collision events of both kinds (we do not want the trajectories of the moving points to be altered by any other events than the linear interpolation) and to increase the numerical stability of the tests.

The result of the introduction of such nonlinear trajectories through linear interpolation was a relatively very drastic dropdown of the overall speed of the application. This is caused by several factors, the first of them is the addition of a new group of events that change the velocity vectors of the points according to their nonlinear trajectories. Additionally, due to the extremely short intervals of linear movement, a very large number of computed events has to be discarded and recomputed (basically repeating the initialization step). The graph in Figure 6.10 shows the overall runtime length of the test for nonlinear point trajectories, note the total amount of time consumed by the test, which is approximately

3.5 times larget than in the linear movement case.

The probably most interesting feature of this graph is the change of the monotony of the displayed curve for the highest percentages of the moving points. This change may possibly be caused by the fact, that the identical phase values force the point moves collinearly for each linear interpolation interval, thus eliminating all the cocircular events, then the vast majority of the runtime would be consumed by the recomputing of the velocity vectors, which may eventually become lower than the runtime consumed by point configuration with slightly lower percentage of the moving points.

6.4 Clustered Data

The concept of clustered data as presented in Chapter 5.7.2 has been implemented with a certain point of success - the cluster movement concept is valid and works as a general idea, however, the computation is weighted by a large amount of numerical imprecision and thus no measurement is possible. These problems are probably caused by the fact that if two moving clusters share some common space, there will be generated fairly large amount of cocircular events in a special kind of singular environment (two sets of points with common velocity vectors). The impact of these drawbacks may be possibly reduced in the future, by introducing some kind of physical model that would prevent the clusters from overlapping.

6.5 Nonplanar Object Simulation

For the following test, we have created a semi-uniform grid of 50×20 points in the bounding are of 100×50 units in size. The points in the triangulation were distributed uniformly except that the x-coordinate of each point that was not inserted into the first row was altered by a random value from interval $\langle -0.5, 0.5 \rangle$ (uniformly distributed). Then the points in the first row (the points with the minimum value of the x-coordinate) were set to move in the direction of the x-axis with the velocity of 5 units per second. All the points in the triangulation were then assigned a height value determined by the Gaussian function as follows:

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \tag{6.1}$$

where $a > 0$ is a real constant that determines the height of the wave (in the case of our application $a = 10$), b is the position of the center of the wave on the x-axis (the x-coordinate of the row of the moving points) and c is a parameter that controls the width of the wave, in our case c is equal to 5. Additional details on the Gaussian function may be found for instance in [29]. A screenshot of the nonplanar object simulation may be seen in Figure A.2 in Appendix A.

6.6 Performance in Other Applications

Together with *Petr Puncman* (see [18]), we have successfully used our implementation of a library for kinetic Delaunay triangulation as a part of a video compression application. Our library provides the application with tools for interpolating the movement of the points between two frames, where the old triangulation is discarded and a new one is created.

The results (presented in the mentioned thesis) show that our method is suitable for this kind of problem, however, the handling of kinetic data must be as fast as possible, which required to speed up our library. The bottleneck of the performance for video compression method lies, quite unexpectedly, in the initialization part of the algorithm (not ain all the cases, because it is dependant on the number of moving points), because it is not unusual to have videos with thousands of points of which only tens are moving. In this case, a very large number of polynomials has to be constructed, but only a certain small part of them will be of a degree greater than zero and thus will have to be solved.

A screenshot from the video compression application may be seen in Figure A.3 in Appendix A. This application uses our implementation of kinetic Delaunay triangulations for compressing video files. The performance of this method is described in [18].

7 Conclusion

The described algorithm provides a suitable way of managing the kinetic Delaunay triangulation in two dimensions. By using the generalized Sturm sequences in combination with the fundamental theorem of algebra, we are able to estimate the locations and multiplicities of all the roots of a polynomial and then, by using the combination of bisection and Newton method, solve the polynomial.

Practical testing showed that the algorithm in its current form is capable of providing a sufficient moving point support for certain applications (the video compression method by *Petr Puncman*), but the overall performance, even although it has been significantly improved since the last version (see [25]), is insufficient. The current capabilities of realtime performance lies somewhere in the order of hundreds of points.

Two possibilities of other types of movement have been considered - a linear interpolation of nonlinear trajectories and the clustered data - but the results obtained from the tests are relatively bad. The numerical errors which accompany the increased number of events in both the mentioned cases, causes instability of the whole triangulation structure. Even though the concept of these types of movement is theoretically sound, the implementation details will need to be worked out. As proved by the nonplanar simulation test, the method is usable for a special sort of pseudo 3D applications.

Tests showed us that the computation of polynomial roots is the most suitable area for further optimization (especially in the case of the polynomials of the fourth degree). The number of the computed and discarded topological events should also be lowered in order to improve the performance.

Bibliography

[1] Gerhard Albers, Leonidas J. Guibas, Joseph S. B. Mitchell, and Thomas Roos. Voronoi diagrams of moving points. *International Journal of Computational Geometry and Applications*, 8(3):365–380, 1998.

[2] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Dewall: A fast divide and conquer Delaunay triangulation algorithm in E^d. *Computer-Aided Design*, 30(5):333–341, 1998.

[3] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry, algorithms and applications.* Berlin Heidelberg: Springer, 1997.

[4] Olivier Devillers. On deletion in delaunay triangulations. In *Symposium on Computational Geometry*, pages 181–188, 1999.

[5] Andrej Ferko. Personal communication, 2007.

[6] Jean-Albert Ferrez. *Dynamic Triangulations for Efficient 3D Simulation of Granular Materials.* PhD thesis, École Polytechnique Fédérale De Lausanne, 2001.

[7] Marina Gavrilova, Jon Rokne, and Dmitri Gavrilov. Dynamic collision detection in computational geometry. In *12th European Workshop on Computational Geometry*, pages 103–106, Munster, Germany, 1996.

[8] Christopher M. Gold and Alfonso R. Condal. A spatial data structure integrating GIS and simulation in a marine environment. *Marine Geodesy*, 18:213–228, 1995.

[9] Holly P. Hirst and Wade T. Macey. Bounding the roots of polynomials. *The College Mathematics Journal*, 28(4):292–295, 1997.

[10] Øyvind Hjelle and Morten Dæhlen. *Triangulations and Applications.* Berlin Heidelberg: Springer, 2006.

[11] Barry Joe. Construction of three-dimensional delaunay triangulations using local transformations. *Comput. Aided Geom. Des.*, 8(2):123–142, 1991.

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

[13] Ivana Kolingerová. *Rovinné triangulace, Habilitační práce.* University of West Bohemia, Pilsen, Czech Republic, 1999.

[14] Ivana Kolingerová. A small improvement in the walking algorithm for point location in a triangulation. In *22nd European Workshop on Computational Geometry*, pages 221–224, March 2006.

[15] Ivana Kolingerová and Borut Žalik. Improvements to randomized incremental delaunay insertion. *Computers and Graphics*, 26(3):477–490, 2002.

[16] Mir Abolfazl Mostafavi, Christopher Gold, and Maciej Dakowicz. Delete and insert operations in Voronoi/Delaunay methods and applications. *Comput. Geosci.*, 29(4):523–530, 2003.

[17] Victor Y. Pan. Solving a polynomial equation: Some history and recent progress. *SIAM Review*, 39(2):187–220, 1997.

[18] Petr Puncman. *Použití triangulací pro reprezentaci videa, Diplomová práce.* University of West Bohemia, Pilsen, Czech Republic, 2008.

[19] Petr Přikryl. *Numerické metody matematické analýzy.* Praha: SNTL, 1985.

[20] Anthony Ralston. *A First Course in Numerical Analysis.* McGraw-Hill, Inc.: New York, 1965.

[21] G. Schaller and M. Meyer-Hermann. Kinetic and dynamic Delaunay tetrahedralizations in three dimensions. *Computer Physics Communications*, 162:9–23, September 2004.

[22] Roman Soukal. *Aplikace algoritmu procházky v počítačové grafice, Diplomová práce.* University of West Bohemia, Pilsen, Czech Republic, 2008.

[23] Roman Soukal. Personal communication, 2008.

[24] David Thibault and Christopher M. Gold. Terrain reconstruction from contours by skeleton construction. *Geoinformatica*, 4(4):349–373, 2000.

[25] Tomáš Vomáčka. Delaunay triangulation of moving points. In *Proceedings of the 12th Central European Seminar on Computer Graphics*, pages 67–74, 2008.

[26] Borut Žalik and Ivana Kolingerová. An incremental construction algorithm for Delaunay triangulation using the nearest-point paradigm. *Int.J. Geographical Information Science*, 17(2):119–138, 2003.

[27] Eric W. Weisstein. Cubic equation. From MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/CubicEquation.html, 2004.

[28] Eric W. Weisstein. Fundamental theorem of algebra. From MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/QuarticEquation.html, 2004.

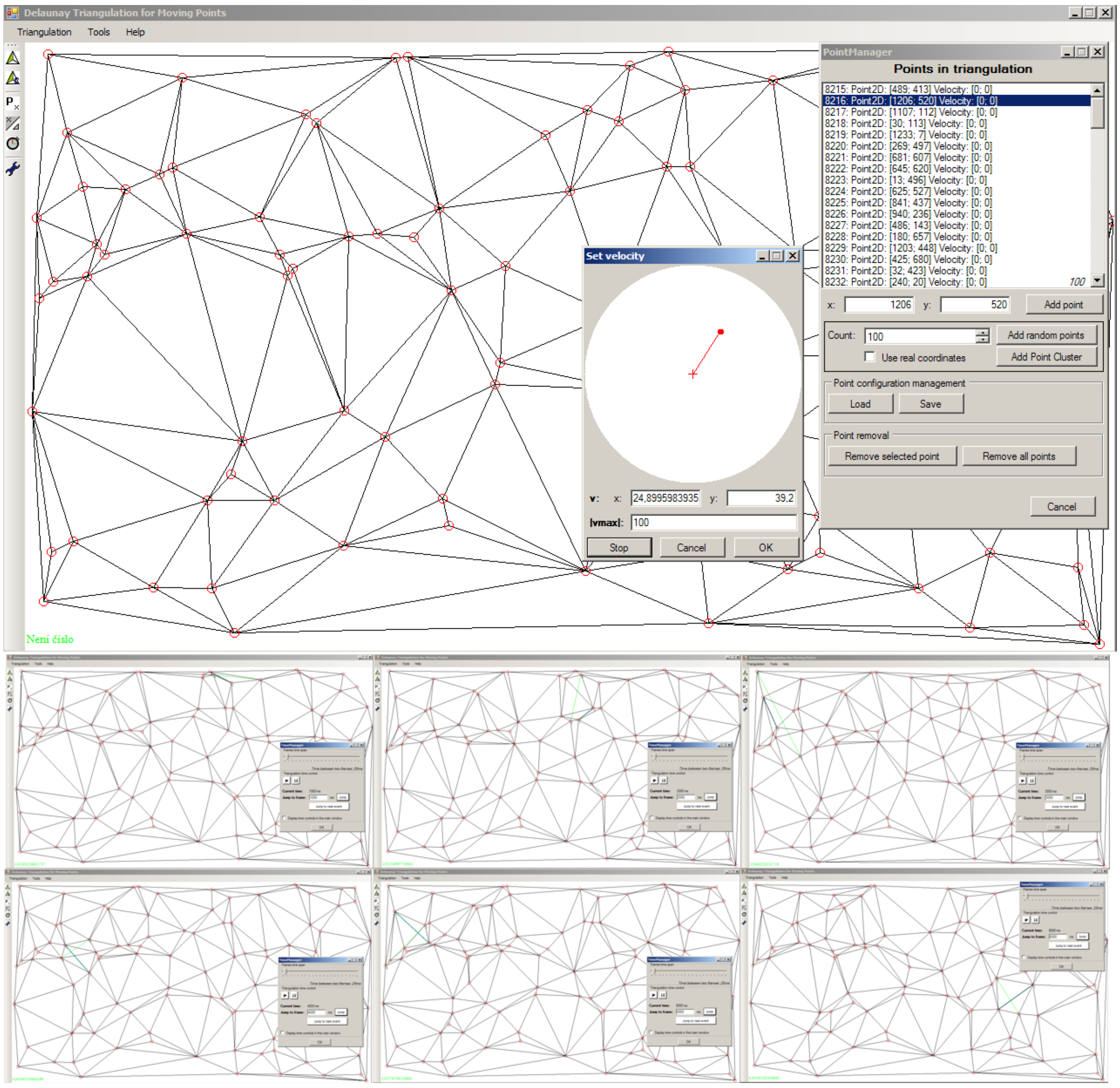
[29] Eric W. Weisstein. Gaussian function. From MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/GaussianFunction.html, 2004.

[30] Eric W. Weisstein. Quartic equation. From MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/QuarticEquation.html, 2004.

Appendices

Appendix A

Screenshots of the Demonstration Applications



```
8215: Point2D: [489; 413] Velocity: [0; 0]
8216: Point2D: [1206; 520] Velocity: [0; 0]
8217: Point2D: [1107; 112] Velocity: [0; 0]
8218: Point2D: [30; 113] Velocity: [0; 0]
8219: Point2D: [1233; 7] Velocity: [0; 0]
8220: Point2D: [269; 497] Velocity: [0; 0]
8221: Point2D: [681; 607] Velocity: [0; 0]
8222: Point2D: [645; 620] Velocity: [0; 0]
8223: Point2D: [13; 496] Velocity: [0; 0]
8224: Point2D: [625; 527] Velocity: [0; 0]
8225: Point2D: [841; 437] Velocity: [0; 0]
8226: Point2D: [940; 236] Velocity: [0; 0]
8227: Point2D: [486; 143] Velocity: [0; 0]
8228: Point2D: [180; 657] Velocity: [0; 0]
8229: Point2D: [1203; 448] Velocity: [0; 0]
8230: Point2D: [425; 680] Velocity: [0; 0]
8231: Point2D: [32; 423] Velocity: [0; 0]
8232: Point2D: [240; 20] Velocity: [0; 0]        100
```

x: 1206 y: 520

Count: 100

[ ] Use real coordinates

v: x: 24,8995983935 y: 39,2

|vmax|: 100

Figure A.1: Managing the kinetic Delaunay triangulation with our demonstration application. Demo application is available on the enclosed CD.

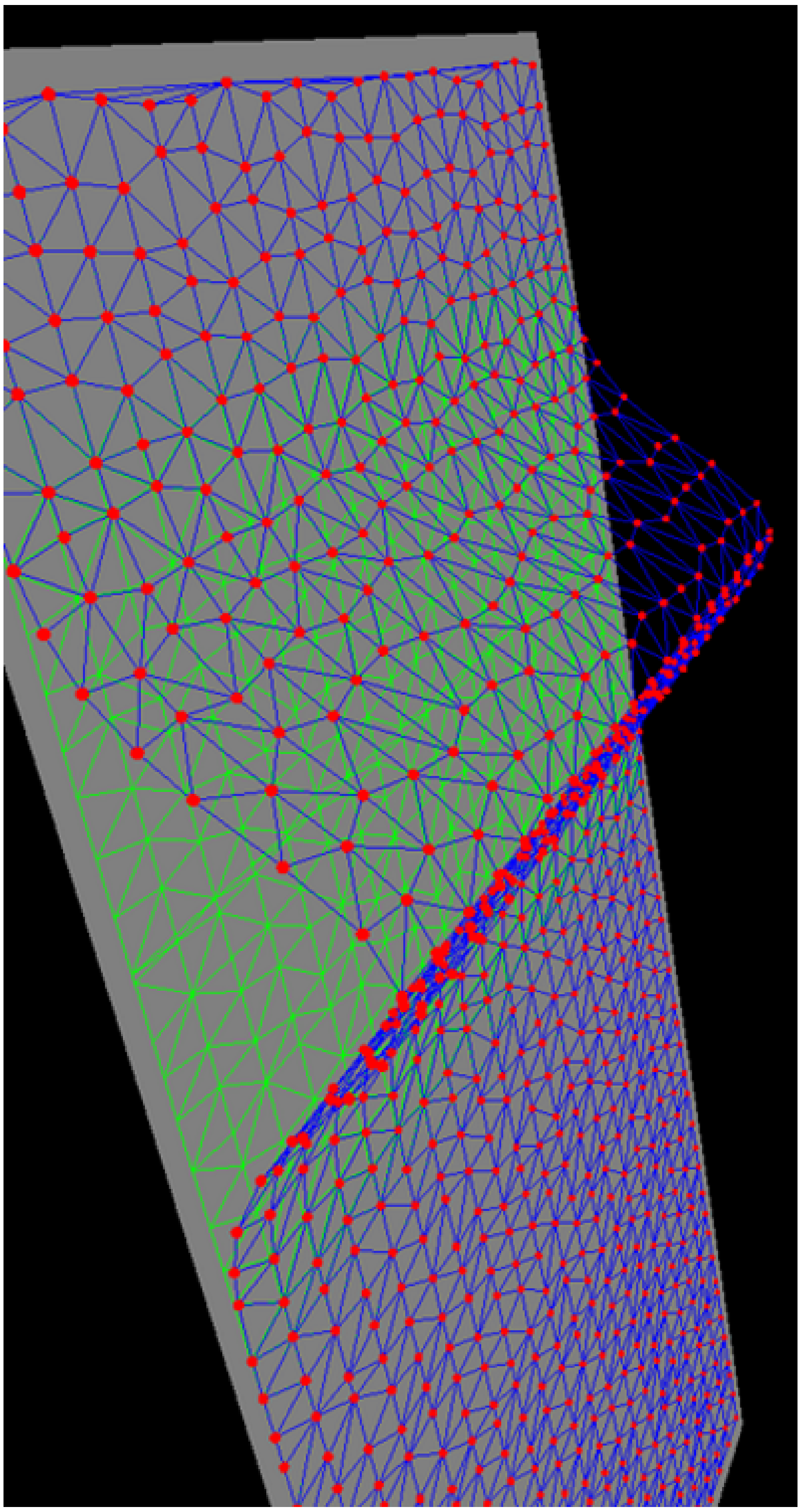
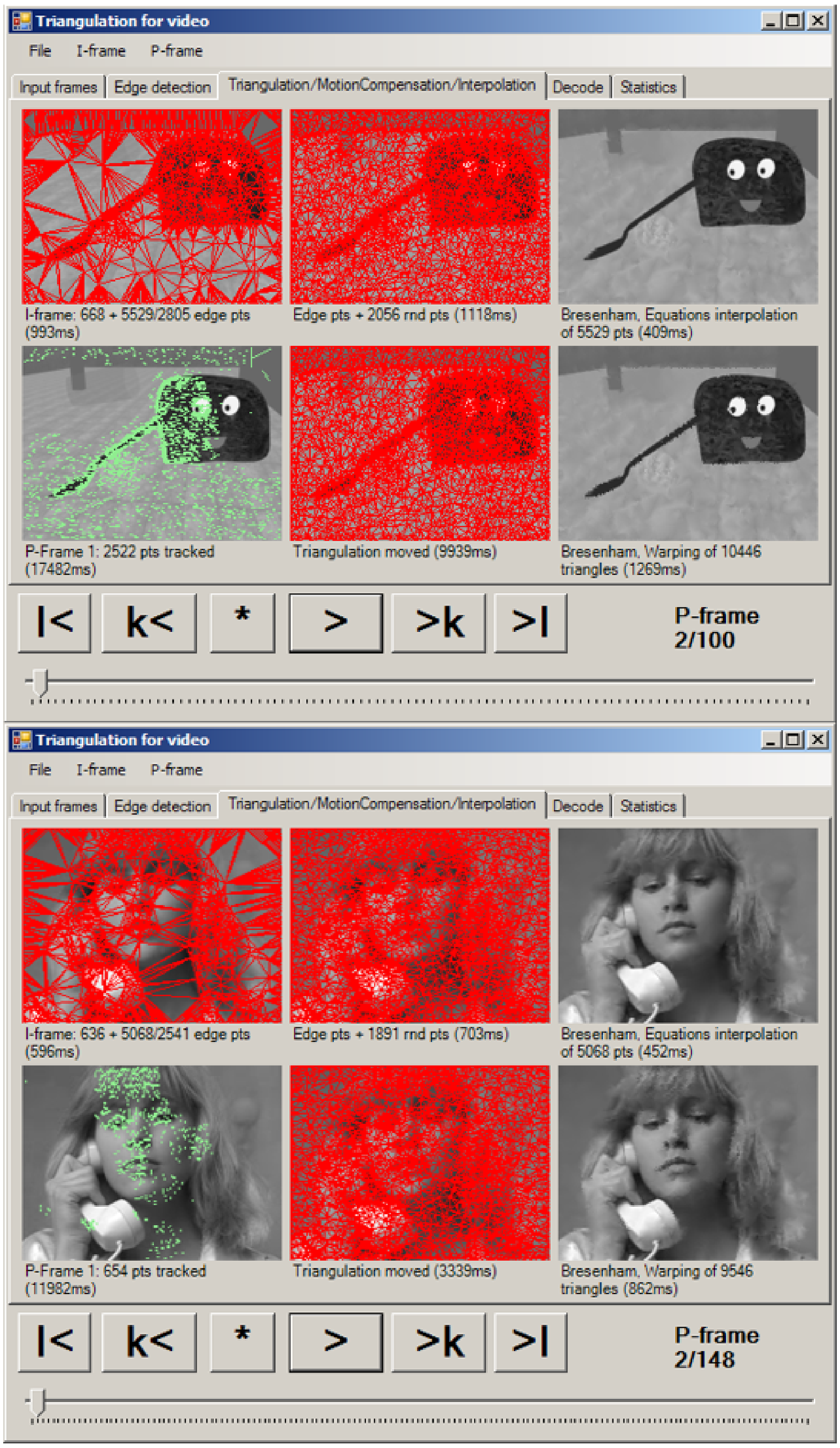


Figure A.2: Simulation of nonplanar object with planar kinetic Delaunay triangulation. Demo application is available on the enclosed CD.



I-frame: 668 + 5529/2805 edge pts (993ms)

Edge pts + 2056 rnd pts (1118ms)

Bresenham, Equations interpolation of 5529 pts (409ms)

P-Frame 1: 2522 pts tracked (17482ms)

Triangulation moved (9939ms)

Bresenham, Warping of 10446 triangles (1269ms)

P-frame 2/100

I-frame: 636 + 5068/2541 edge pts (596ms)

Edge pts + 1891 rnd pts (703ms)

Bresenham, Equations interpolation of 5068 pts (452ms)

P-Frame 1: 654 pts tracked (11982ms)

Triangulation moved (3339ms)

Bresenham, Warping of 9546 triangles (862ms)

P-frame 2/148

Figure A.3: Screenshots from the video compression application by *Petr Puncman*, details may be found in [18].

Appendix B

Využití prioritní fronty pro správu Delaunayovy triangulace nad kinetickými daty

Studentská vědecká konference, 2007

Pilsen, Czech Republic

VYUŽITÍ PRIORITNÍ FRONTY PRO SPRÁVU DELAUNAYOVY TRIANGULACE NAD POHYBLIVÝMI DATY

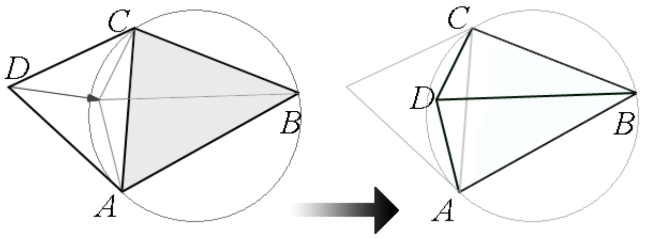
Tomáš VOMÁČKA[1], Ivana KOLINGEROVÁ[2]

1 ÚVOD

Jak ukázala Gavrilova et al. (1996), Delaunayova triangulace představuje díky svým vlastnostem vhodnou datovou strukturu pro detekci kolizí nad pohyblivými daty. Její použití výrazně snižuje algoritmickou složitost problému. Je však třeba zajistit, aby vlivem pohybu bodů nedošlo k tomu, že triangulace přestane být Delaunayovská. Toho lze docílit např. s využitím prioritní fronty pro zpracování topologických událostí, které se v triangulaci vyskytnou. Tato práce zkoumá detekce událostí porušujících triangulaci a jejich zpracování.

2 TOPOLOGICKÉ UDÁLOSTI OVLIVŇUJÍCÍ TRIANGULACI

Vlivem pohybu bodů dochází k porušení Delaunayova kritéria původní triangulace (tj. do kružnice opsané některémtu trojúhelníku vstoupí další bod). Příklad na obr. 1 ukazuje situaci, kdy bod *D* vstupuje do kružnice opsané trojúhelníku *ABC*. Moment, kdy se body *ABCD* vyskytnou na jedné kružnici, nazýváme topologickou událostí.



Obr. 1: Legalizace porušené trojúhelníkové sítě

Detekce tohoto typu topologických událostí probíhá pomocí upravené matice pro test, zda bod leží uvnitř kružnice opsané trojúhelníku.Pro rovnoměrný, přímočarý pohyb bodů,

$$\det\begin{bmatrix} x_{0a}+t\cdot\Delta x_a & y_{0a}+t\cdot\Delta y_a & (x_{0a}+t\cdot\Delta x_a)^2+(y_{0a}+t\cdot\Delta y_a)^2 & 1 \\ x_{0b}+t\cdot\Delta x_b & y_{0b}+t\cdot\Delta y_b & (x_{0b}+t\cdot\Delta x_b)^2+(y_{0b}+t\cdot\Delta y_b)^2 & 1 \\ x_{0c}+t\cdot\Delta x_c & y_{0c}+t\cdot\Delta y_c & (x_{0c}+t\cdot\Delta x_c)^2+(y_{0c}+t\cdot\Delta y_c)^2 & 1 \\ x_{0d}+t\cdot\Delta x_d & y_{0d}+t\cdot\Delta y_d & (x_{0d}+t\cdot\Delta x_d)^2+(y_{0d}+t\cdot\Delta y_d)^2 & 1 \end{bmatrix}=0 \qquad (1)$$

Kde bod $A=[x_{0a};y_{0a}]$ se pohybuje po přímce $(x_{0a}+t\cdot\Delta x_a, y_{0a}+t\cdot\Delta y_a)$ atd. pro body *B, C, D*.

Protože rovnice (1) nabývá tvaru polynomu až 4. stupně, je možné pro její řešení využít analytické metody, Gavrilovou (2003) navrhovanou Newtonovu metodu, metodu dichotomického dělení, popř. pro odhad polohy kořenů i Ralstonem (1973) popsané Sturmovy posloupnosti, nebo kombinaci uvedených metod.

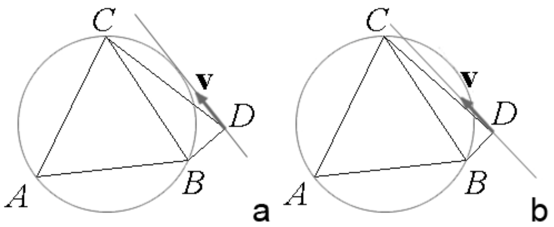
[1] Tomáš Vomáčka, student navazujícího studijního programu Aplikované vědy a informatika, obor Počítačová grafika a výpočetní systémy, e-mail: tvomacka@students.zcu.cz

[2] Doc. Dr. Ing. Ivana Kolingerová, ZČU v Plzni, FAV, Katedra Informatiky a výpočetní techniky, Univerzitní 22, 306 14 Plzeň, tel.: +420 377632433, e-mail: kolinger@kiv.zcu.cz (vedoucí práce)

3 ZPRACOVÁNÍ UDÁLOSTÍ PRIORITNÍ FRONTOU

Po obdržení požadavku na aktuální stav triangulace je potřeba zpracovávat události z vrcholu fronty, dokud neplatí, že čas výskytu první události ve frontě je větší, než aktuální čas triangulace.

Zpracováním události vznikají dva nové trojúhelníky, které je potřeba se všemi jejich sousedy otestovat na nové topologické události. Nové události jsou přidány do fronty na základě polohy a vlastností kořenů rovnice (1). V případě, že získáme kořeny sudé násobnosti, jsou ignorovány, protože značí sudý počet prohození hran, příklad je ilustrován na obr. 2a (dvojitý kořen) a obr. 2b (dva jednoduché kořeny). V případě lichých násobností kořenů určuje nejmenší kořen větší než aktuální hodnota času čas nové topologické události.



Obr. 2: Geometrický význam násobnosti kořenů: (a) dvojitý kořen, (b) dva jedn. kořeny

Mezi nově vznikajícími topologickými událostmi je třeba detekovat tři speciální případy. Trojúhelníky vzniklé prohozením hran jsou testovány na topologické události i vůči sobě navzájem, automaticky tedy získáme informace o nové události v aktuálním čase (právě zpracovávaná událost). Tuto událost je třeba ignorovat a zpracovávat až události následující. Další případy tvoří tzv. zmeškané a falešné události. Zmeškané události představují ty topologické události, které měly být zpracovány, ale buď byly z fronty vyřazeny před svým zpracováním, nebo do ní nebyly vůbec vloženy. Falešné události jsou naopak takové, které byly zpracovány, ačkoliv k tomu nemělo dojít. Společným důsledkem obou těchto jevů je destrukce trojúhelníkové sítě.

4 ZÁVĚR

V rámci řešení dané úlohy bylo třeba prozkoumat metody řešení polynomu 4. stupně. Vzhledem k nízké počáteční informaci o existenci a poloze kořenů se jako optimální ukázala kombinace analytického přístupu a dichotomického dělení.

Důsledkem numerických nepřesností vznikajících během výpočtu v aritmetice s plovoucí řádovou čárkou je degradace prioritní fronty a následná destrukce trojúhelníkové sítě. Při správě triangulace hrají klíčovou roli zmeškané a falešné topologické události, jejichž vzniku je bezpodmínečně nutné zabránit. To bude předmětem další práce.

Poděkování: Děkujeme za poskytnuté informace dr. M. Gavrilové, doc. A. Ferkovi, dr. A. Kolcunovi a prof. S. Míkovi.

LITERATURA

Gavrilova, M., Rokne, J. and Gavrilov, D., 1996. Dynamic collision detection algorithms in computational geometry. *In Proceedings of the 12th European Workshop on Computational Geometry, Munster, Germany.* pp 103-106.

Gavrilova, M., 2003. An Explicit Solution for Computing the Euclidean d-dimensional Voronoi Diagram of Spheres in a Floating-Point Arithmetic. *Computational Science and Its Applications 2669 (3).* pp 827-835.

Ralston, A., 1973. *Základy numerické matematiky.* Academia, Praha.

Appendix C

Delaunay Triangulation of Moving Points

Central European Seminar on Computer Graphics, 2008

Budmerice, Slovakia

Delaunay Triangulation of Moving Points

Tomáš Vomáčka*†

Institute of Computer Graphics
University of West Bohemia
Pilsen / Czech Republic

Abstract

Delaunay triangulation and its dual structure - Voronoi diagram represent a multi-purpose data structures which are widely used in computational geometry. Using these structures for sets of moving data is also relatively well-known and the general approaches have already been discovered. This paper focuses on the rarely discussed problem - computing of the topological events - e.g. the exact times of structural changes in the data structures. Our algorithm uses the Sturm sequences of polynomials to quickly discover the roots, with a possibility to compute only those roots, which are necessary and will most probably be useful.

Keywords: Delaunay Triangulation, Kinetic Data, Computational Geometry

1 Introduction

According to [6], Delaunay triangulation of moving points represents an efficient way of collision detection. It is so because a point has to be checked for collision only against its neighbors in the triangulation (e.g., with all the points to which it is connected) and only when a new edge containing this point as a vertex is added to the triangulation.

Even though the collision detection is the most straightforward (and most often discussed) application of the Delaunay triangulation of moving points, it is by far not the only one. Together with the transformation to Voronoi diagram it may provide a base data structure for path planning in an environment with moving objects (especially when extended to three dimensional space). Some triangulation-based methods for video compression may take advantage of the triangulation that changes its structure according to the movement of the points. Other use of similar data structures may be easily found anywhere the data represent moving points.

There are two main approaches to moving the points in a triangulation. The first of them (and the simpler one) is to remove each moving point and then reinsert it back to the triangulation at new coordinates. This approach has a significant disadvantage when we want to use it for a collision detection - when a point moves relatively fast, its new position may be so far from the original one that some edge insertion and removal may be skipped. If this edge represents a collision edge (e.g., this fast moving point collides with the other vertex of this edge), a collision will be missed.

The other approach models continuous movement and utilizes a priority queue to keep track of scheduled topological events. When a request is made to acquire the current state of the triangulation, events from the queue are popped and processed until the inner time of the triangulation matches the requested one. This procedure ensures that the triangulation structure will only be altered when the movement of the points causes a topological change.

This paper focuses only on points moving with constant velocity vector. This limitation may seem too serious, but the mathematical relations described in this paper may be (with some effort) modified for movement along polynomial curves. Other types of trajectories cannot be generally solved in the same way and are beyond the focus of this paper.

Known and described techniques of solving the problem are described in Section 2. Section 3 of this paper provides some basic definitions. Inner structural changes of the triangulation as a result of the movement of the points are described in Section 4. Section 5 describes several ways of obtaining the topological events and outlines geometrical meaning of the solved equations with an emphasis on the count and multiplicity of their roots. Section 6 documents results of our work so far. Summary of the project and further work proposals are given in Section 7.

2 State of the Art

Even though various papers on similar subject propose the technique discussed in this paper (see [6, 7]), almost nothing has been written about obtaining the topological events from the mathematical description of the movement of the points. However the principle of this approach, as well as the general iteration algorithms for maintaining the structure of kinetic Delaunay triangulations or Voronoi diagrams, is well known and described together with the theoretical bounds of number of the processed topological events in [1]. Even non-Euclidian metrics such as the

*tvomacka@students.zcu.cz

†This work was supported by Ministry of Education - project No. LC06008

power and Manhattan metrics have been considered for this problem, see [5], where those metrics are applied on a set of moving discs and line segments.

Each of the mentioned articles describes the process of obtaining the topological events (discussed later) as a problem of finding real roots of the 4-th order polynomial. This polynomial cannot be in practice solved analytically (although the relations are known), so numerical solutions are suggested (with almost no details on which numerical methods should be used and why).

Although some online software libraries for polynomial solving exist (for instance the GSL library - see [3]), the implemented algorithms used for polynomial solving are usually based on the analytical approach or optimized for finding the complex roots of polynomials. Both of these options are unsuitable for our work.

We propose a new algorithm which determines the amount, approximate location and multiplicity of the roots and which allows us to simply discard some of the roots and enumerate the others.

3 Definitions

3.1 Triangulation

Triangulation $T(S)$ of a set of points S in the Euclidean plane is a set of edges E such that

- no two edges in E intersect at a point not in S,
- the edges in E divide the convex hull of S into triangles

Delaunay triangulation $DT(S)$ over a finite set S of n points in $2D$

$$S = \{P_1, P_2, ...P_n\}$$

is the triangulation that fulfills the condition that no point is inside the circumcircle of any triangle in $DT(S)$. This property, known as the Delaunay condition, is a key feature in our application and must be preserved over time despite the movement of the points.

To determine if a triangle $P_1P_2P_3$ and a point P_4 satisfy the Delaunay condition, the incircle test must be made over the three points of the triangle and the considered point. If $P_i = [x_i, y_i]$ where $x_i, y_i \in \mathbb{R}$ represent the coordinates of points $P_1, ..., P_4$, then we can determine the position of P_4 against the circumcircle of the triangle $P_1P_2P_3$ according to the sign ot the determinant of the matrix $\mathbf{I}$ (for details see [8]):

$$\det \mathbf{I} = \det \begin{pmatrix} x_1 & y_1 & x_1^2+y_1^2 & 1 \\ x_2 & y_2 & x_2^2+y_2^2 & 1 \\ x_3 & y_3 & x_3^2+y_3^2 & 1 \\ x_4 & y_4 & x_4^2+y_4^2 & 1 \end{pmatrix} \qquad (1)$$

If the vertices of the triangle $P_1P_2P_3$ are oriented counter-clockwise, then the positive sign of Eq. (1) means that P_4 lies inside the circumcircle of $P_1P_2P_3$, negative sign means that P_4 lies outside and zero always means (independently on the orientation of the vertices of the triangle) that P_4 lies exactly on the circumcircle.

3.2 Point Movement

Points $P_1, ..., P_n$ are moving at a constant velocity and their coordinates must be thus defined as linear functions of time:

$$\begin{aligned} P_i(t) &= [x_i(t), y_i(t)] & (2) \\ x_i(t) &= x_{i0} + \Delta x_i \cdot t, y_i(t) = y_{i0} + \Delta y_i \cdot t & (3) \end{aligned}$$

where $t \geq 0$, $P_i(0) = [x_{i0}, y_{i0}]$ is the initial position of the point P_i, e.g. the position of its insertion and $\Delta x_i, \Delta y_i \in \mathbb{R}$ represent velocity coordinates of P_i. We require the initial positions of the points to be inside a triangulation area - a rectangle in E^2 defined as:

$$O = < x_{min}; x_{max} > \times < y_{min}; y_{max} >$$

and state that no point may ever leave this rectangular area. If a point is to move outside the given bounds, a collision event will occur and (as described in Section 3) change the velocity of the point in such a fashion to keep it inside the boundaries.

3.3 Priority Queue

A priority queue is an abstract data type, which provides the following operations:

- Push (i, t): add the item i to the queue with respect to the priority t.
- Pop: remove item i with the highest priority from the queue and return it.
- And sometimes others, such as returning the first element in the queue without removing it (known as the "Head" function).

4 Triangulation Behavior

4.1 Overall Functionality

Functionality of the algorithm (also described in [1, 5]) may be divided in two steps - the preprocessing and the iteration. In the preprocessing step, the Delaunay triangulation of the points in their initial positions is created (in our case by using the Incremental Insertion algorithm - see [2]) and the first topological event is computed for each pair of adjacent triangles by determining the nearest time their four points become cocircular (see further). In addition to those events, the collision times are computed for each pair of points connected with an edge, forming point-point collision events, and the collision times for each point with the boundaries of the triangulation area (let the edges of the bounding rectangle be known as the walls), forming point-wall collision events.

All the computed events are then placed into the priority queue with the priority defined as:

$$p = t_{curr} - t_{event}$$

where $t_{event} \in \mathbb{R}$ is the time of the execution of the event and $t_{curr} \in \mathbb{R}$ is the current time of the triangulation $(t_{event} \geq t_{curr})$.

The iteration step is repeated each time a request for the triangulation state is received. If the current time of the triangulation is lower than the current time, the event from the head of the queue is popped and executed (this may lead to adding some new events to the queue as well as removing some of the events in the queue) and the current time of the triangulation is set to the time of the executed event. This step is repeated until the current time of the triangulation matches the requested time.

4.2 Explanation of Topological Events

When the triangulation contains at least one point with a nonzero velocity vector, its structure will have change in time due to the Delaunay condition. As shown in [6, 1], moving points may change their position without structural changes in the triangulation until a topological event occurs (see Figure 1). As shown in the figure, the topological event occurs when four points become cocircular and it is thus determined by the time a point (point $P_4 \rightarrow P_4' \rightarrow P_4''$ here) enters a circumcircle of a triangle $P_1P_2P_3$. At this point the triangulation becomes non-Delaunay. At this time, the Delaunay condition is violated and the triangulation must be repaired by processing the topological event.

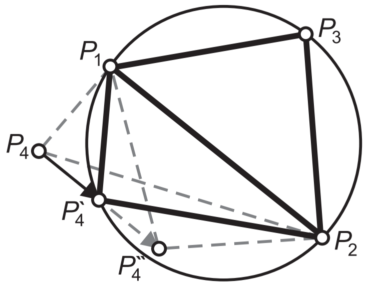


Figure 1: Triggering of the topological event

4.3 Creating the Topological Events

When all points are added into the triangulation and movement starts, topological events are scheduled for each edge in triangulation. For each edge e we test its vertices for mutual collision and collision with the walls of the bounding rectangle and then, if e is shared by two triangles, we compute the nearest topological event for their four points in the future (events in the past may be byproduct of the computation and are discarded for obvious reasons).

If any of the performed computations result in a positive event time greater than the current time of the triangulation, we store it in the priority queue. This means that point-point and point-wall collisions are stored in the queue along with topological events and are processed similarly (see further). Figure 2 shows a simple example of queueing the events. As we can see, the point P_4 moves with the velocity vector $\mathbf{v}_4$ and this movement will cause at least four events displayed in the Queue box of Figure 2.

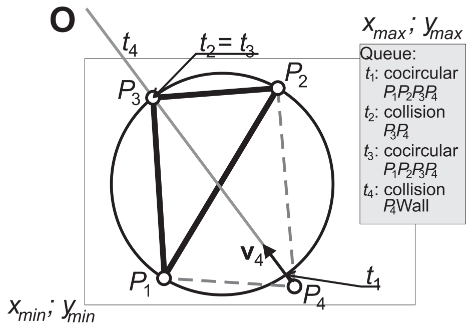

O
t_4
$t_2 = t_3$
$x_{max}; y_{max}$
Queue:
t_1: cocircular $P_1P_2P_3P_4$
t_2: collision P_3P_4
t_3: cocircular $P_1P_2P_3P_4$
t_4: collision P_4Wall
P_3
P_2
$\mathbf{v}_4$
P_1
t_1
P_4
$x_{min}; y_{min}$

Figure 2: Scheduling the events for $P4$

Note that the collision event in the time t_4 determines the time when P_4 will leave the area O. Proper handling of this type of events will ensure that all points will remain inside the boundary. Also note the fact that collisions between points represent singular cases and are always time identical with cocircular events. These singular cases may be handled in various ways. For instance safety disc may be added around each point or the events. The safety discs only serve for computing collision times - they make two points collide when they come close enough, creating collision events in situations where they would not otherwise occur. They also force the points to collide earlier than they would without them if the collision would occur anyway and thus eliminate the singularities. Another way to handle the singular cases is to order events in the priority queue in such way that if two events of a different kinds are scheduled to the same time, then a collision event should be executed before any cocircular event. This precaution helps in situations where one point is deflected away from another one by reactive forces (these forces are of course dependent on implemented physical model), without changing the topology of the triangulation. For any other time near the collision, there is only one legal configuration and it is the original one, so the edge swap is not necessary.

4.4 Processing the Topological Events

When a topological event is triggered, the triangulation structure changes. As mentioned in [6, 1], the changes will be local - to process a topological event means to swap the common edge (see Figure 3) of the two triangles involved in the event and schedule new topological events.

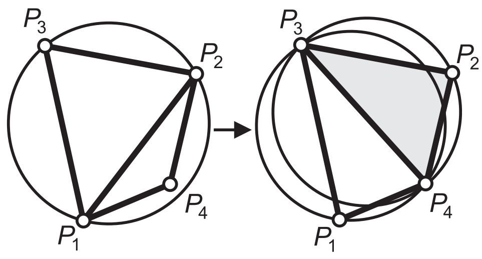


Figure 3: Edge swap as a result of a topological event

Along with scheduling topological events, vertices of the new edge (generated by the edge swap) must be tested for mutual collision. Removing triangles from the triangulation (which is a result of swapping the edges) makes some events in the queue invalid, because the considered triangle do not exist anymore. This fact must be considered and the invalid events must either be removed from the queue immediately or discarded when popped. Algorithm in Figure 4 shows the complete procession of a topological event.

5 Computing the Topological Events

5.1 Basic Relationships

To determine time of a topological event, we have to compute the time, when four points become cocircular. This can be done by solving the Eq. (4), where $\mathbf{I}$ denotes the incirlce-test matrix from relation (1), with point coordinates defined as in (2) and (3). This equation is a modified version of the incircle test which is normally used for constructing Delaunay triangulations.

$$\det \mathbf{I} = 0 \tag{4}$$

In this equation, coordinates $[x_1, y_1], ..., [x_4, y_4]$ represent time dependent coordinates of points P_1, P_2, P_3 and P_4 as defined in Section 3.2. If the coordinates of the points are linear functions of time, then solving this equation means to solve a polynomial of the fourth or lower degree.

The determinant of $\mathbf{I}(t)$ will not change if we substract the first row of $\mathbf{I}(t)$ from all its rows. This transformation means we set the first point to be identical with the origin. Using this technique, we transform the fourth row of $\mathbf{I}(t)$ to $[0,0,0,1]$, but the maximum order of the solved polynomial remains unchanged and equal to four.

Due to the fact that we are only interested in topological events taking place in the future, we do not have to search for all the roots of the equation. We just have to obtain the roots which are greater than or equal to the current time of the triangulation.

Input:

- Ev - the topological event on top of the priority queue; $Ev.T_1$, $Ev.T_2$ - the involved triangles
- Let $Ev.T_1 = P_1P_2P_3$ and $Ev.T_2 = P_1P_4P_2$ as in Figure 3

Output:

- Update of the topological structure of the triangulation and events in the priority queue.

Auxiliary:

- Q – priority queue
- DT – Delaunay triangulation of the points $P_1, ..., P_n$

Algorithm:

```
• $Ev \leftarrow Q.pop()$
• if ($Ev.T_1$ is invalid or $Ev.T_2$ is invalid)
    – discard $Ev$ and exit
• Swap the common edge of $Ev.T_1$ and $Ev.T_2 \rightarrow$
  $Ev.T_1 = P_1P_4P_3$ and $Ev.T_2 = P_2P_3P_4$
• Test $P_3$ and $P_4$ for point collision $Col$ at time $t_{Col}$
    – $Q.push(Col, t_{Col})$
• For each triangle $N$ sharing a common edge with $Ev.T_1$
    – Test $Ev.T_1$ and $N$ for the nearest future topological event $Ev_1$ at time $t_{Ev1}$
        * $Q.push(Ev_1, t_{Ev1})$
• For each neighbor $N$ ($N \neq Ev.T_1$) of $Ev.T_2$
    – Test $Ev.T_2$ and $N$ for nearest future topological event $Ev_2$ at time $t_{Ev2}$
        * $Q.push(Ev_2, t_{Ev2})$
```

Figure 4: Processing of a topological event

5.2 Polynomial Root Dependency on Nature of Topological Events

As told before, count and multiplicity of roots of equation (1) depends on which points are moving and how. For instance, when only one point (of the four considered points) moves, there is no possibility that the polynomial will have more than two roots (or one double root). This is because the velocity vector of this moving point and its current position define a line and the three other points define a circle. By solving the given equation, we are looking for the points of intersection of the line and the circle. The following figures show some of the basic examples of root dependency on the positions and velocities of the involved points.

Figure 5 shows the situation when a point P_4 moves tangentially to the circumcircle of the triangle $P_1P_2P_3$. In this

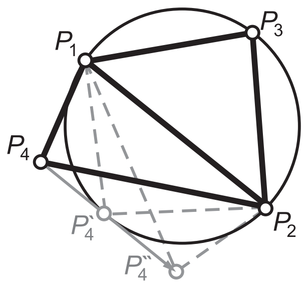


Figure 5: Tangential movement of the point P_4

case we will obtain one double root by solving Eq. (4). This fact means that two edge swaps are taking place at the same time. By swapping the edge even times (i.e. twice or four times in our case) we return the two triangles to their original state. This means that when we search for topological events, we can ignore all roots of even degree.

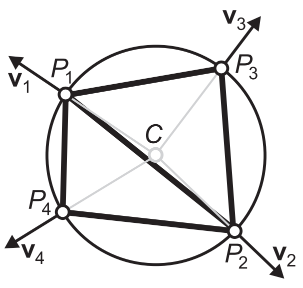


Figure 6: Points moving away from the center of their circumcircle

When the four points P_1, P_2, P_3 and P_4 are cocircular and moving with velocity vectors $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ and $\mathbf{v}_4$ as shown in Figure 6 (the center of their circumcircle C lies on all four of their movements' trajectories), then two singular cases may occur:

1. $\mathbf{v}_1 = \mathbf{v}_2 = \mathbf{v}_3 = \mathbf{v}_4 = \mathbf{0}$
   In this case, all the points are cocircular and not moving. The equation (1) degenerates into $0 = 0$ and cannot be solved.

2. $\|\mathbf{v}_1\| = \|\mathbf{v}_2\| = \|\mathbf{v}_3\| = \|\mathbf{v}_4\| \neq 0$
   The points move away from their circumcenter equally fast. This means that there will be a topological event for each $t \in \mathbb{R}$ as the circumcircle will grow. We may discard all of the obtained topological events because both possible triangle configuration are legal due to all four points lying on the same circle and thus no edge swapping is necessary. Similar situation arises when the points are all moving towards their circumcenter.

Other special point configurations may exist, but the ones mentioned in Figures 5 and 6 are highly valuable for solving the topological event equation.

5.3 Approaches to solving of the equation

As mentioned before, in order to obtain the topological events, we need to solve a polynomial of the fourth or lower degree. Let us define the solved polynomial as in Eq. (4). We can enumerate the coefficients of $p(t)$ by transforming Eq. (1) to more suitable form ($\det \mathbf{I}(t) \equiv p(t)$).

$$p(t) = \sum_{i=0}^{4} a_i \cdot t^i = 0 \tag{5}$$

where $a_i \in \mathbb{R}$.

Theoretically, the given polynomial may be solved analytically using Vieta's and Cardano's formulas (see [11] for solving quartic equations and [10] for solving cubic equations). However, the limited floating point precision and subresults being complex numbers make this approach inadvisable. These features result in both unprecise results and an incorrect number of roots (including their multiplicity). Quadratic and linear equations may be solved analytically with sufficiently precise results.

Various numerical methods represent another option for finding the roots of the polynomial, Newton's method does not represent a good option because small values of the solved polynomial derivation may cause the method to find next iteration very far from the current one and possibly converge to a different root. Another downside of the Newton's method presented in [9] is the fact that it has problems in finding roots of multiplicity greater than one. A better method proposed by [4] and described in [9] is called Sturm Sequences.

$$\begin{aligned} f_1(x) &= f(x) \\ f_2(x) &= f'(x) \\ f_{j-1}(x) &= q_{j-1}(x) f_j(x) - f_{j+1}(x), j = 2, ..., m-1 \\ f_{m-1}(x) &= q_{m-1}(x) f_m(x) \end{aligned} \tag{6}$$

As proved in [9], a sequence of polynomials in Eqs. (6) is Sturm sequence. Eqs. (6) also show the way of construction of Sturm sequence from a polynomial $f(x)$. 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 (in our case this sequence will have no more than four terms). The most important feature of Sturm sequence of polynomials in our case is the fact that it allows us to easily determine the count of real roots in any interval $\langle a; b\rangle$ (a or b may be even infinite) and determine their multiplicities. Note that only the remainders of each division have to be counted, the quotients are not needed in further steps of the construction of the sequence.

To obtain the root values, we only have to count $V(a) - V(b)$ where the function $V(x), x \in (R)$ determines the number of signum changes between successive polynomials in the sequence (zeros are ignored). Multiplicities of the roots may be easily determined by solving the last polynomial in the sequence. As proved in [9], each multiple root of $f_1(x) = f(x)$ with the multiplicity $r > 1$ is also a root of $f_N(x)$ with multiplicity equal to $r-1$. Here, m denotes the count of polynomials in the sequence. Considering the fact that in our case $f_m(x)$ is a polynomial of the third or lower degree and that the total count of complex roots of any polynomial with real coefficients must be even, we can formulate the guidelines to solving the polynomial[1] as presented in Table 1.

$\deg f(x)$	$f_m(x)$ real root mult.	$f(x)$ real root mult.
3	{2}	{3}
3	{1}	{2, 1}
3	none	{1} or {1, 1, 1}
4	{3}	{4}
4	{2}	{3, 1}
4	{1, 1}	{2, 2}
4	{1}	{2} or {2, 1, 1}
4	none	{1, 1} or {1, 1, 1, 1}

Table 1: Features of the polynomial depending on its Sturm sequence

Input:

- $p(t) = \sum_{i=0}^{n} a_i \cdot t^i = 0$ ... a polynomial of degree n

Output:

- Sequence $\{t_i\}_{i=1}^{r}$ of the real roots of $p(t) = 0$, $r \leq n-1$. Or empty sequence, if no real roots exist.

Algorithm:

```
• if($n \leq 2$)
    – Compute analytically, return the sequence of roots $\{r_0, ..., r_n\}$.
• if($n = 3$)
    – Solve using the *Sturm3* algorithm - see Figure 8
• if($n = 4$)
    – Solve using an extension to the fourth degree of polynomials of the *Sturm3* algorithm from Figure 8. It is not listed in this paper, because the idea is the same as in the third order algorithm.
```

Figure 7: Computing the roots of a polynomial

If a polynomial $f(x)$ has at least one multiple root x_i of multiplicity r, we can divide it by polynomial $(x - x_i)^r$ and thus decrease its order. The result of this division may then be solved analytically, because in the worst case the original polynomial $f(x)$ is of the fourth degree and the root x_i of multiplicity two. By dividing a quartic polynomial by a quadratic one, we get another quadratic polynomial as a result. If degree of $f(x)$ is three or four and it has no multiple roots, we solve its derivate (processing recursively for the third order polynomial as a derivate of the fourth order polynomial) and thus obtain all the local extremes of $f(x)$. Local extremes then define intervals that bound roots of $f(x)$. From these intervals we may enumerate the roots using some iteration method (such as - in the simplest case - bisection). The whole procedure is shown by the algorithm in Figure 7.

[1]We only consider cases where degree of $f(x)$ is greater than two, because linear and quadratic equation may be solved analytically as told before. Also if the polynomial has no roots, we do not attempt to solve it.

Input:

- $p(t) = \sum_{i=0}^{3} a_i \cdot t^i = 0$ - a third order polynomial

Output:

- Sequence $\{t_i\}_{i=1}^{r}$ of the real roots of $p(t) = 0$, $r \leq 3$. Or empty sequence, if no real roots exist.

Auxiliary:

- Sturm sequence $f_1(t), ..., f_m(t)$ of the polynomial $p(t)$ - see Eqs. (6), note that $f_1(t) = p(t)$.

Algorithm:

```
• Create Sturm sequence for $p(t)$.
• $r_{count} \leftarrow (V(-\infty) - V(\infty))$
  $V(x),\ x \in (R)$ determines the number of signum changes between successive polynomials in the sequence (zeros are ignored)
• if($r_{count} = 0$)
    – Return empty sequence of roots {}
• $R_m = \{r_{mi}\}_{i=1}^{r_{mult}} \leftarrow$ sequence of $r_{mult}$ roots of $f_m(t)$ (e.g. the multiple roots of $p(t)$)
• if($\|R_m\| = 2$)
    – Return $\{r_{m1}, r_{m1}, r_{m1}\}$ (one triple root)
• if($\|R_m\| = 1$)
    – $p(t)$ has a double and a single root (see Tab. 1).
    – $r_s \leftarrow$ the only single root of $\frac{p(t)}{(t-r_{m1})^2} = 0$
    – Return $\{r_{m1}, r_{m1}, r_s\}$ (a double and a single root)
• else
    – Solve $p(t)$, using a suitable numerical method.
    – Return $\{r_i\}_{i=1}^{r}$ ... sequence of $r \leq 3$ distinctive roots.
```

Figure 8: *Sturm3* algorithm

6 Performance

Presented results were obtained from a *C#* implementation of the discussed algorithms. Our primary goal is the creation of a robust and stable program, speed optimization has not been introduced yet. All presented results were obtained for a random configuration of 100 points with a safety discs of 1 unit diameter in 1000×1000 units rectangle. Certain percentage of the points was moving in a random direction and velocity. Program performance was observed during a 10 second interval and the final results represent average values for three different sets of points.

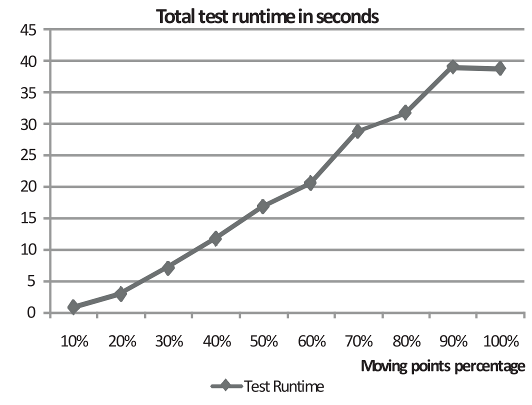


Figure 9: Total runtime needed for the test.

The graph in Figure 9 shows the dependency of the total runtime needed for the execution of the whole test on the percentage of the moving points. Assuming from the measured values, the needed runtime has time complexity with upper bound of $O(n^2)$ and with lower bound of $O(n)$.

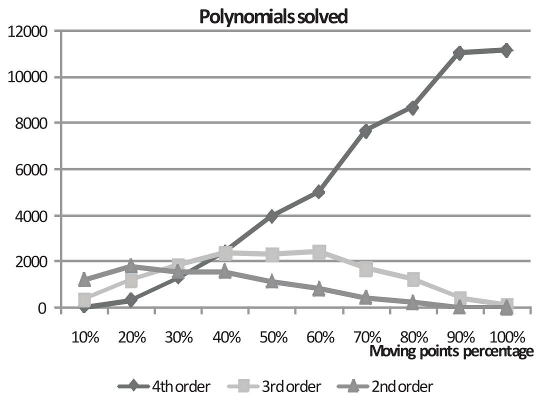


Figure 10: Number of polynomials solved during the program life cycle

Figure 10 presents the dependency of the number of polynomials of different orders solved during the whole life cycle of the program. As we can see from this graph, the number of the fourth degree polynomials grows with the percentage of the points that are moving. The number of third order polynomials has a global maximum for 50% moving points ratio and is at near-zero value for 0% and 100% moving points ratio. The second order polynomials form the majority for low percentages of moving points but their number decreases for higher moving points ratios. This behavior is caused by the fact that the degree of polynomial in Eq. (4) generally grows for increasing number of non-static points in the configuration of two adjacent triangles. It is less likely to count topological events for triangle pairs with three or four moving points in the configurations with the lower percentages of moving points.

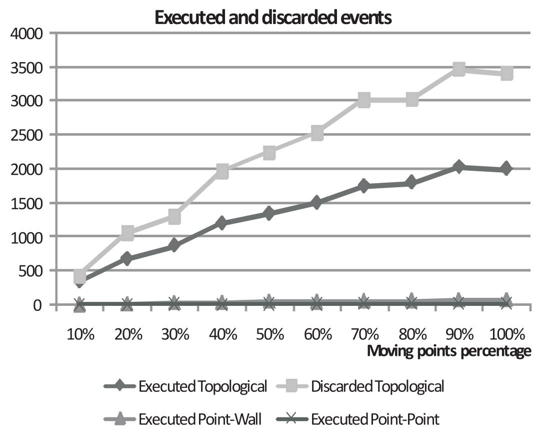


Figure 11: Numbers of executed and discarded events of various kinds during the life cycle of the program

Algorithm in Figure 4 shows that some of the topological events are discarded due to the topological changes in the triangulation structure. Graph in Figure 11 shows the numbers of executed and discarded topological events, as well as the numbers of executed events of the other types. We can see that the count of topological events (both executed and discarded) is much greater than the counts of the events of the other types. Another remarkable fact is that the number of discarded topological events is always greater than the number of the executed topological events. There seems to be an upper bound of $O(n)$ and a lower bound of $O(\log n)$ on the number of both executed and discarded topological events.

Another consequence of the behavior demonstrated by the graph in Figure 10 is shown in the graph in Figure 12 - runtime spent on solving of the polynomials (of both the third and the fourth degree) represents a vast majority of time spent during the life cycle of the program. The other parts of the program consume less than 10% of the runtime for approximately 30% and greater moving point percentages. This is caused solely by the increasing number of solved polynomials because runtime needed to solve one polynomial remains constant. Most of the time consumed by solving polynomials in the current version of the program is needed to numerically enumerate the roots.

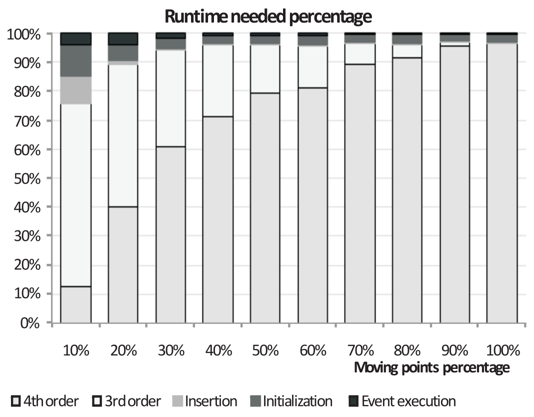


Figure 12: Runtime needed for the main parts of the program - the 4th order polynomial solving, the 3rd order polynomial solving, insertion of the points into the triangulation, the initialization step of the movement and total time spent on the event execution

7 Conclusion and Future Work

We presented a new algorithm for determining the time of the topological events. Our algorithm provides a hybrid numerical-analytical way of solving the polynomials of the fourth or lesser degree with sufficient precision.

Future improvement of the performance of the algorithm is possible. Results obtained by the tests determine the polynomial solving part of the algorithm as the most suitable area for further optimization (for example by initiating some highly sophisticated numerical method). Another possibility of speeding up the performance lies in the minimization of the number of discarded topological events. If the redundant events were successfully recognized, the corresponding polynomials would not have to be solved at all.

Our algorithm is currently being used as a part of a triangulation-based video compression program developed at the Institute of Computer Graphics of the University of West Bohemia. Future usage of our algorithm involves path planning and collision detection applications. Extension to 3D and considering other types of point movement represent another possibilities of further development.

Acknowledgement

This work would not be created without the advice and patient guidance of Dr. I. Kolingerová from the University of West Bohemia, Pilsen, Czech Republic, to whom I would like to thank. My thanks also belong to Mr. A. Kolcun from the Academy of Sciences of the Czech Republic, to Dr. A. Ferko from Comenius University, Bratislava, Slovakia and to Dr. M. Gavrilova from the University of Calgary, Calgary, Canada for their insight into the problem and helpful ideas.

References

[1] Gerhard Albers, Leonidas J. Guibas, Joseph S. B. Mitchell, and Thomas Roos. Voronoi diagrams of moving points. *International Journal of Computational Geometry and Applications*, 8(3):365–380, 1998.

[2] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry, algorithms and applications*. Berlin Heidelberg: Springer, 1997.

[3] M. Galassi et al. Gnu scientific library reference manual (2nd ed.). From GSL - GNU Scientific Library. http://www.gnu.org/software/gsl/.

[4] Andrej Ferko. Personal communication, 2007.

[5] Marina Gavrilova and Jon Rokne. Swap conditions for dynamic voronoi diagrams for circles and line segments. *Comput. Aided Geom. Des.*, 16(2):89–106, 1999.

[6] Marina Gavrilova, Jon Rokne, and Dmitri Gavrilov. Dynamic collision detection in computational geometry. In *12th European Workshop on Computational Geometry*, pages 103–106, Munster, Germany, 1996.

[7] Ignacy R. Goralski and Christopher M. Gold. Maintaining the spatial relationships of marine vessels using the kinetic voronoi diagram. In *ISVD*, pages 84–90. IEEE Computer Society, 2007.

[8] Øyvind Hjelle and Morten Dæhlen. *Triangulations and Applications*. Berlin Heidelberg: Springer, 2006.

[9] Anthony Ralston. *A first course in numerical analysis*. McGraw-Hill, Inc.: New York, 1965.

[10] Eric W. Weisstein. Cubic equation. From MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/CubicEquation.html, 2004.

[11] Eric W. Weisstein. Quartic equation. From MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/QuarticEquation.html, 2004.