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]):

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

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:

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

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.

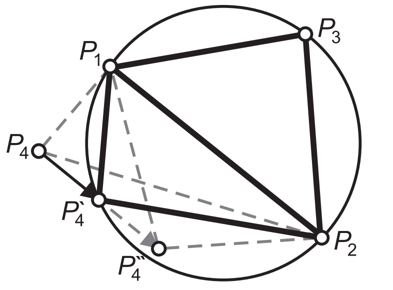

Figure 1: Triggering of the topological event

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.

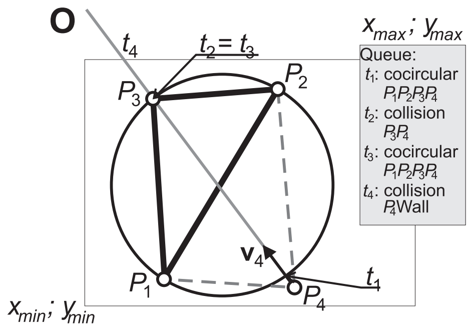
O
x_{max}; y_{max}
t_4
$t_2 = t_3$
Queue:
t_1: cocircular $P_1P_2P_3P_4$
t_2: collision P_3P_4
t_3: cocircular $P_1P_2P_3P_4$
t_4: collision P_4Wall
P_3
P_2
$\mathbf{v}_4$
P_1
t_1
P_4
x_{min}; y_{min}

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.

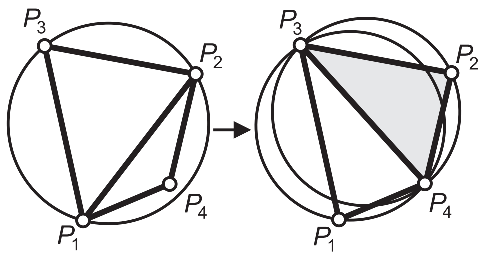

Figure 3: Edge swap as a result of a topological event

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

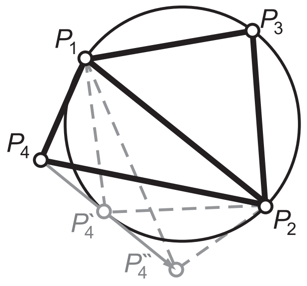

Figure 5: Tangential movement of the point P_4

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.

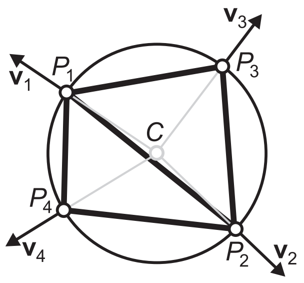

Figure 6: Points moving away from the center of their circumcircle

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.

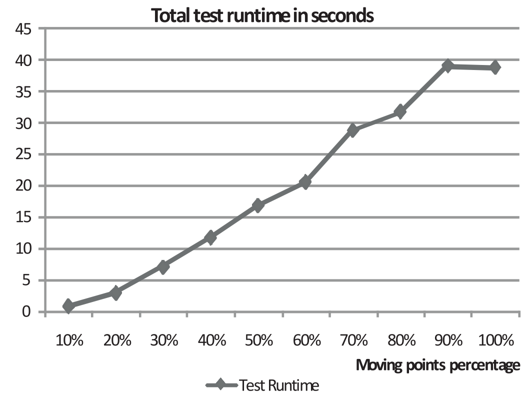

Figure 9: Total runtime needed for the test.

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 $O(n)$.

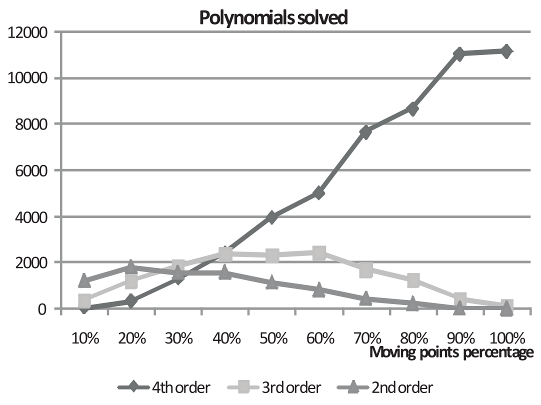

Figure 10: Number of polynomials solved during the program life cycle

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.

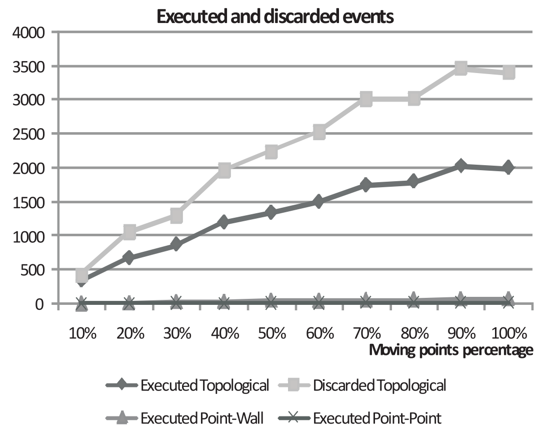

Figure 11: Numbers of executed and discarded events of various kinds during the life cycle of the program

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.

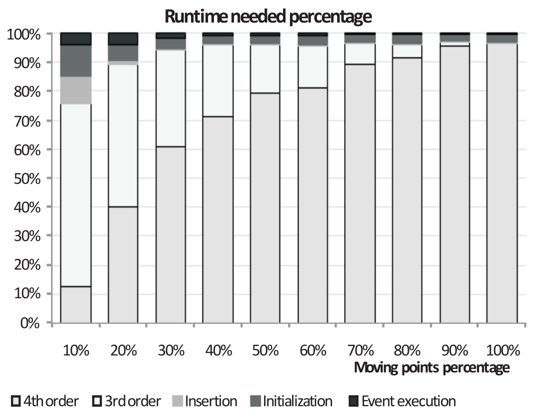

Figure 12: Runtime needed for the main parts of the program - the 4th order polynomial solving, the 3rd order polynomial solving, insertion of the points into the triangulation, the initialization step of the movement and total time spent on the event execution

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.