Computation of Topologic Events in Kinetic Delaunay Triangulation using Sturm Sequences of Polynomials

Tomáš Vomáčka*
Ivana Kolingerová†
University of West Bohemia

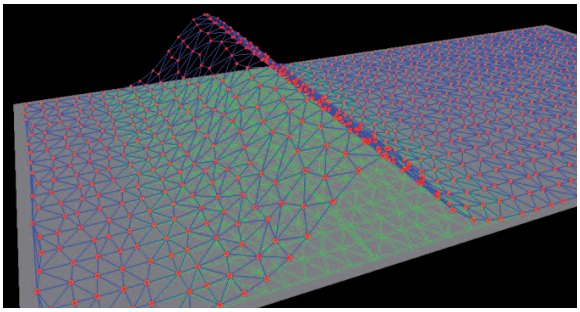

Figure 1: *Experimental implementation of the kinetic Delaunay triangulation for 2.5D wave simulation.*

Abstract

Even though the problem of maintaining the kinetic Delaunay triangulation is well known, this field of computational geometry leaves several problems unsolved. We especially aim our research at the area of computing the times of the topologic events. Our method uses the Sturm sequences of polynomials which, combined together with the knowledge in associated field of mathematics, allows us to separate the useful roots of the counted polynomial equations from those which are unneeded. Furthermore, we adress the problem of redundant event computation which consumes an indispensable amount of the runtime (almost 50% of the events is computed but not executed). Despite the deficiency in the fields of speed and stability of our current implementation of the algorithm, we show that a large performance enhancement is theoreticaly possible by recognizing and not computing the redundant topologic events.

CR Categories: G.1.5 [Numerical Analysis]: Roots of Nonlinear Equations—Polynomials, methods for; I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Geometric algorithms, languages, and systems

Keywords: Kinetic Delaunay Triangulation, Polynomial, Sturm Sequence

1 Introduction

Delaunay triangulation (DT) constructed over a set of time-dependent data (also called kinetic DT) represents a multi purpose data structure that may be used in various fields of computer science. The most commonly discussed use of kinetic DT is collision detection in environments with moving obstacles - see [Gavrilova et al. 1996] - because certain features of DT ensure that there will be an edge between each two points that are about to collide and thus we only have to test for collision the pairs of points which are connected by an edge of the DT. Another example of a similar application of kinetic DT is path planning in similar kinds of environment. In this case the triangulation is transformed to its dual Voronoi diagram which is then used to find the most suitable path. An example of such an application is for instance the marine vessel navigation as described in [Gold and Condal 1995]. We have also used the kinetic DT as a base data structure for video representation. In this application the movement of the points in the triangulation describes the correspondency between two consecutive frames of the video sequence and the triangulation is reinitialized upon reaching a keyframe. More information on this topic may be found in [Puncman 2008; Vomáčka 2008c]. Another example of possible use of kinetic DT are virtual reality environments. The kinetic DT may then serve for the purpose of communication with the user who then determines the movement of the kinetic data by the actions he or she takes inside the virtual world.

This paper focuses on the problem of maintaining a kinetic Delaunay triangulation with points that move along linear trajectories with no acceleration. We use the continous movement approach with the topologic events being stored in a priority queue. Furthermore, we focus on the analysis of the computation of the times of these events and propose some enhancement for the method we use for the computation.

This paper is organized in the following fashion: Section 2 describes the known and most often used approaches for solving the given problem. Section 3 provides the necessary definitions. Section 4 focuses on the computation and handling of the topologic events. Section 5 introduces the Sturm sequences of polynomials to the reader and shows how they are used in the process of solving the polynomials. Section 6 describes the results we achieved with our implementation of the described algorithm. Section 7 summarizes the whole paper. A preliminary version of the method has already been presented in [Vomáčka 2008a]. The performance and stability of the method has been strongly improved since the ver-

*e-mail: tvomacka@kiv.zcu.cz
†e-mail: kolinger@kiv.zcu.cz ; Work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research project LC-06008 (Centre for Computer Graphics).

sion presented there. The enhancements in both of those fields are in detail presented in [Vomáčka 2008b; Vomáčka 2008c] and this paper describes only their most essential parts. However, additional possibilities of further improving the stability and performance are given in this work.

2 State of the Art

Two main approaches for managing DT of moving data have been developed. The first of them separates the movement into discrete time instants and then utilizes repeated removal of the moving points from the triangulation followed by reinserting these points back into the triangulation at new coordinates. Such a triangulation is then often referred to as a fully dynamic DT. This approach has some obvious disadvantages resulting from the discrete understanding of the time of the moving objects. The possibly most important one of them is the fact that this approach may not be used in quite a large area of applications (such as collision detection) due to the risk of missing some important events that may occur between two consecutive discrete times. This situation may occur for instance in the case that some points in the triangulation are moving relatively fast and their new position is thus relatively far away from their original coordinates. On the other hand, this approach is extremely simple and requires no additional computation besides the triangulation construction (in this case, those algorithms which are online are considered to be the best ones - see [Vomáčka 2008b]) and point removal. It is also unaffected by the type of movement of the points, which may be extremely useful in certain types of applications. Algorithms for removing points from Delaunay triangulation are described for instance in [Devillers 1999; Mostafavi et al. 2003] and many others. A detailed review of the dynamic data structures including the use of this approach for movement simulation may be found in [Mostafavi et al. 2003]. Another example of this method is presented in [Thibault and Gold 2000].

The second approach is based on the idea that the topology of the triangulation changes only at certain time instants called topologic events - see [Gavrilova et al. 1996]. It has been shown that these events occur only when four points in the triangulation become cocircular as a result of their movement. Between two consecutive topological events, all the points in the triangulation may move without any restrictions (except the type of their trajectory which has to be linear in the vast majority of the cases). The algorithms which utilize this approach have to solve two important problems - how to obtain the times of the topologic events and how to handle these events. In the case that the triangulation is constructed using Euclidean metrics and the points move along linear trajectories with no acceleration, the event times may be found by solving polynomial equations of degree up to four - see [Albers et al. 1998]. The events may then be stored in a priority queue with the priority being the time of their occurrence thus allowing us to update the state of the triangulation by popping events from the queue, processing them and eventually pushing new ones back into the queue as they are computed during the process. Another way to process the topologic events is to split the continuous time of the triangulation into very short intervals as described in [Ferrez 2001] and postpone the evaluation of the events during each interval to its end. This can, however, lead to some numerical inconveniences and in the case of collision detection even to missing a collision, but if the intervals are short enough, the triangulation state will be legal at the end of each interval. Another method that can be used for evaluating the movement of the points is described in [Schaller and Meyer-Hermann 2004] as a method for point removal. It is computationaly very simple but may only be used when only one point is moving at a time.

It is also common that the papers which use the above mentioned continuous movement approach (see above) only state that the polynomials have to be solved but give no information on what is the best way of solving them. These polynomials cannot be in practice solved analytically due to floating point arithmetics imprecision. On the other hand, from all the available methods for solving polynomials (or nonlinear equations in general), only few are suitable for the discussed problem because of their unnecessary complexity or their focus on finding all of the complex roots of the given polynomial which is both unnecessary and unwanted (introducing the complex numbers into this kind of computation may increase the degree of imprecision). Overview of these methods (which include, but are not limited to, Lehmer-Schur method, Bairstow method, Bernoulli's method and others) may be found for instance in [Pan 1997; Ralston 1965].

3 Definitions

3.1 Triangulation and Delaunay 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 in 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 the Euclidean plane

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

is the triangulation that fulfills the condition that no point $S_i \in S$ is inside any triangle in $DT(S)$ - see [Hjelle and Dæhlen 2006]. This property is sometimes also called the Delaunay criterion or the empty circumcircle criterion. Because this condition determines the Delaunay triangulation, it is thus crucial that it is preserved during the whole lifetime of the algorithm despite the point movement. To determine if a point and a triangle satisfy this criterion a matrix test is performed - it is often called the incircle test and is described further.

3.2 Incircle Test

Let us have a triangle $P_1P_2P_3$ and a point P_4, to determine whether the point lies inside, on or outside the circumcircle of the triangle we have to compute the determinant of the following matrix $\mathbf{I}$:

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

where $P_i = [x_i, y_i]$ are the coordinates of the point P_i in the Euclidean plane.

By computing the value of $\det \mathbf{I}$ we can determine the mutual position of the point against the circumcircle of the triangle. For instance, if the triangle $P_1P_2P_3$ is oriented counterclockwise and $\det \mathbf{I} > 0$ then point P_4 lies inside the circumcircle of the given triangle and value of $\det \mathbf{I} = 0$ always means that the four points are cocircular, despite the orientation of the triangle. Detailed information on the incircle test may be found in [de Berg et al. 1997]

3.3 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:

$$P_i(t) = [x_i(t), y_i(t)] \quad (2)$$
$$x_i(t) = x_i^0 + \Delta x_i \cdot t \quad (3)$$
$$y_i(t) = y_i^0 + \Delta y_i \cdot t$$

where $t \geq 0$ is the current time of the triangulation (i.e., the time since the movement has started) and $P_i(0) = [x_i^0,\ y_i^0]$, $x_i^0,\ y_i^0 \in \mathbb{R}$ is the initial position of point P_i, i.e., the position at which it was inserted into the triangulation and $v_i = [\Delta x_i, \Delta y_i]$ is the velocity vector of point P_i.

Additionally we require each of the points to have its initial position (the position for $t = 0$) to be inside a certain triangulation area - a rectangle in Euclidean plane defined as:

$$\mathbf{O} = \langle x_{min}, x_{max} \rangle \times \langle y_{min}, y_{max} \rangle \quad (4)$$

where $x_{min}, x_{max}, y_{min}, y_{max} \in \mathbb{R}$ are the boundaries of the triangulation area. No point may then leave area $\mathbf{O}$ and if it is about to move outside of the given bounds, a collision will occur and change the velocity of the point in such a fashion to keep it inside the boundaries.

3.4 Topologic Event

As shown in [Albers et al. 1998; Gavrilova et al. 1996] and others, if the triangulation contains at least one point with nonzero velocity vector, its structure will have to change in time in order to keep the Delaunay condition valid. The moving points may change their position freely until they reach such a position when the change is inevitable and topologic event occurs - see Fig. 2.

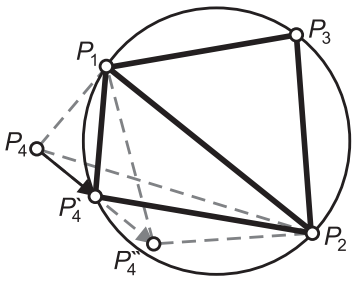

Figure 2: *Triggering of a topologic event.*

As shown in this figure, the topologic 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 because the Delaunay condition is violated and the triangulation must be repaired by processing the topological event which is done by swapping the edge common to the two triangles in question - i.e., triangles $P_1P_2P_3$ and $P_1P_4'P_2$ here become $P_1P_4'P_3$ and $P_2P_3P_4'$ (illustration is given in Fig. 3, where point P_4 has moved inside a circumcircle of the triangle $P_1P_2P_3$. The resulting edge swap then changes the local topology as displayed in the figure). It is also important that the change in the topology of the triangulation is local, as shown in [Gavrilova et al. 1996], however more than one event may occur at the same time.

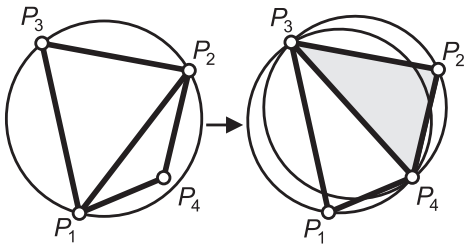

Figure 3: *Processing of a topologic event.*

4 Topologic Event Handling

4.1 Topologic Event Computation

In order to compute the time of a topologic event, we have to determine the time when four points that define two adjacent triangles become cocircular. This can be done by solving an equation created by establishing Eq. (3) into Eq. (1):

$$\det \mathbf{I}(t) = \det \begin{bmatrix} x_1(t) & y_1(t) & x_1^2(t) + y_1^2(t) & 1 \\ x_2(t) & y_2(t) & x_2^2(t) + y_2^2(t) & 1 \\ x_3(t) & y_3(t) & x_3^2(t) + y_3^2(t) & 1 \\ x_4(t) & y_4(t) & x_4^2(t) + y_4^2(t) & 1 \end{bmatrix} = 0 \quad (5)$$

where $x_i(t)$ and $y_i(t)$ are time dependent coordinates of the points in the triangulation as defined in Eq. (3). Because the coordinates of the moving points are linear functions of time (we consider them to move along linear trajectories with no acceleration), we can see that to solve Eq. (5) means to solve a polynomial equation of degree four or less depending on how many of the points in the two adjacent triangle configuration are actually moving.

4.2 Using the Priority Queue

As the points are inserted into the triangulation, the first future topology event is computed for each pair of adjacent triangle (if such exists) and these events are then placed into a priority queue. This process is often called the initialization step of the kinetic DT - see [Vomáčka 2008a]. From this queue the events are then popped in the order in which they occur and are executed. The process of executing a topologic event consists of swapping the common edge of the two triangles that defined the event as shown above. And because of the fact that two triangles are removed from the triangulation and are replaced by two new ones, new topologic events may occur and some of the previously computed events may become illegal. If any new events are computed, they are pushed into the priority queue and all of the now illegal events are removed from it without executing them. The *pop-execute-push* cycle is then repeated as needed in order to preserve the Delaunay condition of the triangulation and is called the iteration step.

4.3 Redundant Event Computation

It is vital to note that many of the computations of topologic events are redundant - some of the events will have to be computed more than once or will be computed but will never be processed. Let us consider the situation displayed in Fig. 4 and 5. In order to make the problem simpler to observe, the points in the figures that describe this problem are moved subsequently (the point P_4 remains static during the movement of P_5 and vice versa). From the definition of our approach, the points should move simultaneously. In that

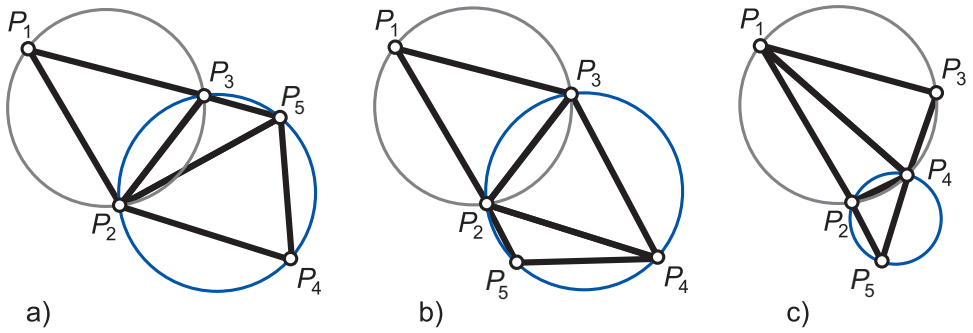

Figure 4: *Topologic changes in the vicinity of an upcoming event.*

case the speed of movement of P_4 would be much slower than the speed of P_5. The consecutive movement of points reffers to both the Figures 5 and 4.

As we may see, the topologic event for triangles $P_1P_2P_3$ and $P_2P_4P_3$ will occur at the time t_0. Another topologic event is scheduled for triangles $P_2P_4P_3$ and $P_3P_4P_5$ and it will occur at the time t_1. Even another topologic event will occur for the same pair of triangles at the time t_2 but it is not stored in the queue yet because we only push the first future topologic event for each triangle pair. Please note that events which are taking places at the time t_1 and t_2 are computed with respect to the movement of the points P_4 and P_5 and thus will occur at the circumcircles of the triangles that do not exist yet (their topology is correct but the position of the points will change due to the movement). In order to keep the figure as simple as possible, these events are marked on the currently existing circumcircle which will change its radius due to the movement of point P_4.

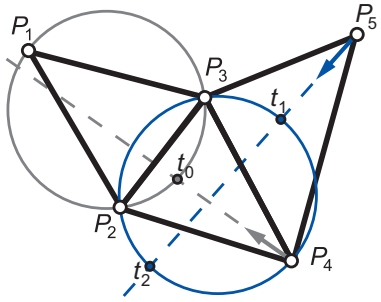

Figure 5: *Redundant computatation of topologic events.*

Let us then suppose that $t_1 < t_2 < t_0$. The events will occur in the following order: t_1, t_2, t_0. In this case the topology of the triangulation changes at t_1 as shown in Fig. 4a, making the event scheduled for triangles $P_1P_2P_3$ and $P_2P_4P_3$ at t_0 illegal and removing it from the queue. After that, the topology changes again at t_2 (Fig 4b), reverting partialy to the original state. This change enforces the computation of a topologic event for the triangle pair $P_1P_2P_3$ and $P_2P_4P_3$ which results in the topology event at the time t_0 which has been already computed and has been discarded. The situation can be even worse when multiple points leave and enter the vicinity of a triangle pair similar to the one displayed in Fig. 5.

In the case that $t_1 < t_0 < t_2$ (the order of the occurence of these events is that the event at t_1 will occur as the first one, supposedly followed by the event at t_0 and then by the event at t_2). The situation is similar to the previous case - the first topologic event, scheduled at t_1 makes the event at t_0 illegal and it is then removed from the queue. But due to the fact that $t_0 < t_2$, this exact topology event will never occur because the triangulation does not return to its original topologic state (or if it does so it is not soon enough for the original event to be replaced by its exact copy as in the previous case).

4.4 Singular Cases

As mentioned before, the times of topologic event are obtained by computing a polynomial equation of degree up to four in the form described in Eq. (5). The events (if some exist) are then denoted by the roots of this equation. The location and multiplicity of the roots may then give us some vital information whether and how should they be processed. The most obvious use of this information is the fact that we generally want to use only those roots that take place in the future and of them usualy only the first ones. So if the current time of the triangulation is equal to $t = t_{curr}$ and we obtain roots of t_1, t_2, t_3, t_4 as a result of solving the equation in the form of the one shown in Eq. (5) then if $t_1 < t_2 < t_{curr} < t_3 < t_4$, the roots t_1 and t_2 are obviously out of our focus and we only push into the queue the event that takes place at t_3. This principle has, however, some less obvious but maybe even more important use. It allows us to recognize singularities caused by the movement of the point which must be solved in a special fashion. One example of such a singularity is a tangential movement of a point towards a circumcircle of an adjacent triangle. An illustration of such a case is given in Fig. 6. Other types of singularities are described in [Vomáčka 2008a; Vomáčka 2008b].

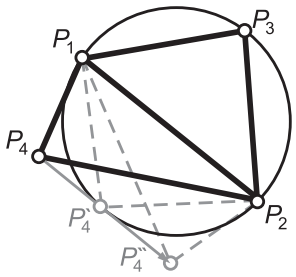

Figure 6: *Tangential movement of a point against a circumcircle.*

What we see in Fig 6 is that the point $P_4 \rightarrow P_4' \rightarrow P_4''$ moves tangentially towards the circumcircle of the triangle $P_1P_2P_3$. At the time it reaches the position marked as P_4', both of the available triangular configurations are legal (i.e., the edge swap may occur, but it is not necessary). However, for each other position of P_4, only the displayed triangle configuration is legal. This feature is a result of the fact that the root obtained by solving the appropriate

equation is of double multiplicity and thus two edge swaps should occur. Knowing that two edge swaps result in exactly the same configuration as was the one we started with, we may as well skip both of them. The same rule applies to each root of even multiplicity - see [Vomáčka 2008b].

5 Sturm Sequences of Polynomials

5.1 Mathematical Requirements

Because of very high expected amount of computed polynomial equations (an equation will be computed for each pair of adjacent triangles and one equation will be computed per each executed topologic event), it would be useful if we had a mathematical tool that would help us to compute only such roots of these equations that we really need and can use. The Sturm sequences of polynomials, which are defined later in this section, allow us to do this by splitting the domain of the solved polynomials into suitable intervals, each of which contains exactly one root (ignoring the multiplicities of these roots). This feature allows us to take full advantage of the fact described earlier in Section 4.4 that we are only interested in the future topologic events (and usually even only in the nearest one of them). Combined together with the ability to separate the roots of a polynomial, we are able to run the time-consuming iterative computations only on a relatively small portion of the interval we would otherwise have to search.

The Sturm sequences of polynomials also enable us to detect (and possibly avoid) the singular cases mentioned above. These singularities are recognized by even multiplicities of the roots which denote the topologic events in question or degenerated cases of the polynomials (for instance $0 \cdot t = 0$). One of the most important features in the singular case detection is the fact that they may be easily detected immediately after the Sturm sequence is created. The detection is possible because the roots of the last term of the newly constructed Sturm sequence directly corresponds with the multiple roots of the original polynomial (see later).

5.2 Definition

As defined in [Ralston 1965], the sequence of polynomials

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

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

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

   $\forall t \in \mathbb{R} : f_k(t) = 0 \Rightarrow f_{k-1}(t) f_{k+1}(t) < 0, k = 2, ..., m-1$

5.3 Construction

Sturm sequence of a polynomial $f(t)$ may be constructed (as proved in [Ralston 1965]):

$$\begin{aligned} f_1(t) &= f(t) \\ f_2(t) &= f'(t) \\ f_{j-1}(t) &= q_{j-1}(t) f_j(t) - f_{j+1}(t), j = 2, ..., m-1 \\ f_{m-1}(t) &= q_{m-1}(t) f_m(t) \end{aligned} \tag{6}$$

In these relations, $q_{j-1}(t)$ is the quotient and $f_{j+1}(t)$ is the negation of the remainder of division of the polynomial $f_{j-1}(t)$ by the polynomial $f_j(t)$. $\{f_i(t)\}$ 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 $f_i(t)$ 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. (6) to the following form:

$$\frac{f_{j-1}(t)}{f_j(t)} = q_{j-1}(t) + (-1) \cdot \frac{f_{j+1}(t)}{f_j(t)} \tag{7}$$

What we see in Eq. (7) is a division of two polynomials, with $f_{j-1}(t)$ being the numerator and $f_j(t)$ being the denominator of the division. $q_{j-1}(t)$ then denotes the quotient (which is unused for the creation of the Sturm sequence) and $f_{j+1}(t)$ is the negation of the remainder of the division (the multiplication of $f_{j+1}(t)$ by the constant -1 is necessary to make the relation mathematically correct).

5.4 Important Features and Generalization

Counting the Roots Let us define a function $V(t)$ as the count of the number sign changes in the Sturm sequence as in Eqs. (6) (ignoring all zeros). This function may then be used to count the number of distinct real roots of $f(t)$ on any interval $\langle a, b\rangle$:

$$r_{\langle a,b\rangle} = V(a) - V(b) \tag{8}$$

where $a, b \in \mathbb{R}$ or either of a, b may be infinite. As proved in [Ralston 1965], Eq. (8) remains valid even if a or b are the roots of $f(t)$.

Root Multiplicity The last term of the Sturm sequence as in Eqs. (6) may be used to distinguish and compute the values of the multiple roots of $f(t)$. As proved in [Ralston 1965], all the multiple roots of $f(t)$ with multiplicities decreased by one are the roots of $f_m(t)$, 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 $f_m(t)$ is of an odd degree, then $f(t)$ 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 Section 5.2, 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 $\{f_i(t)\}$ by multiplying all of its terms by any polynomial $p(t)$, thus gaining a sequence in the form of $\{p(t) \cdot f_i(t)\}$. If a Sturm sequence is mentioned anywhere in the following text, a generalized Sturm sequence is meant.

5.5 Solving the Polynomials Using Sturm Sequences

The process of solving a polynomial (in order to obtain the times of topologic events) consists of several steps (we only solve polynomials of degree greater than two because the linear and quadratic equations may be solved analytically with sufficient precission.). At first, a Sturm sequence is constructed for the given polynomial. We then attempt to solve its last term in order to find the multiple roots of the original polynomial. If there are any of them, we divide the original polynomial by $(t - t_i)^m$, where t_i is the multiple root and m is its multiplicity and we proceed with the result of this division to find the remaining roots. If there are none multiple roots, we solve the derivate of the original polynomial in order to estimate the

intervals which contain its roots and then use a suitable numerical method to locate them (for instance the Newton method). The reason why this process is possible is that the number and multiplicities of roots of a real polynomial are determined by the fundamental theorem of algebra - see [Weisstein 2004]. This theorem determines that the number of complex roots of a real polynomial is either even or zero and thus allows us to limit the possibile combinations of the number and multiplicities of the roots of a polynomial to very small number which can further be reduced to one available combination by solving the derivate polynomial of the original polynomial equation.

The whole procedure of solving a polynomial of the third degree is sumarized in Algorithm 1, solving a polynomial of higher degrees is very similar to the described one. Additional details on this method may be found in [Vomáčka 2008b].

Algorithm 1: 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. (6), note that $f_1(t) = p(t)$.
- $R_m = \{r_m^i\}_{i=1}^{r_{mult}}$ - a sequence of all the multiple roots of $p(t)$. Each multiple root r_m^i 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 Eq. (8)`

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 degrees.
```
    Return an 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
```
    $r_s \leftarrow$ 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 $r \in \{1, 3\}$ distinct roots.
end

6 Results

6.1 Theoretical Results

In order to examine the theoretical expectations, we have implemented the algorithm described in this paper and run several tests. The most important of their results are presented in this section. All the tests described here were performed on random sets of points generated by using the following approach: 100 random points were placed in a bounding area 100×100 units in size. These points each had 1 unit in diameter safety disc (two points collide if their become at least as close as is the sum of the radii of their safety discs). Then certain percentage of the points were assigned a random velocity vector with each of its component being a random number from $\langle -5; 5 \rangle$ interval. This configuration was then observed for 10 seconds for different percentages of the moving points.

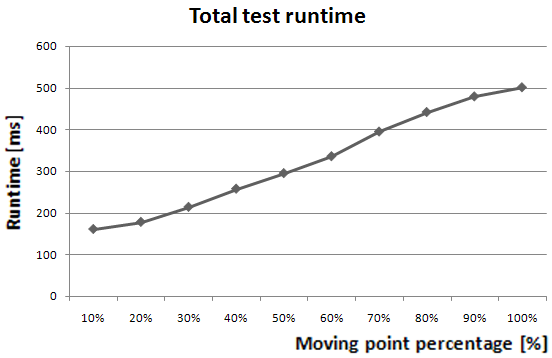

Figure 7: *Total runtime consumed by the test.*

Figure 7 shows the total runtime consumed by the tests as described above. According to the measured values, we may assume that the consumed runtime has an upper bound of $O(n)$ and a lower bound of $O(\log n)$.

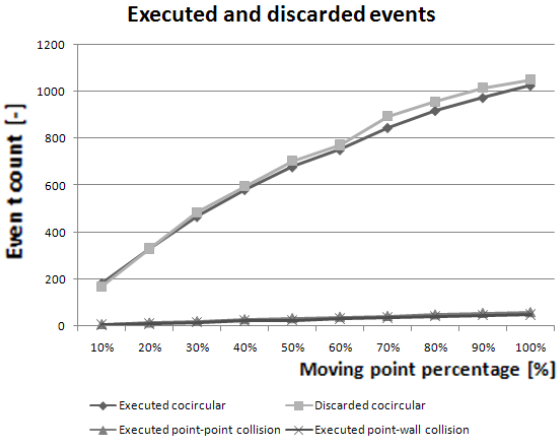

Figure 8: *Numbers of executed and discarded events of various types.*

In Fig. 8 we can see that very large number of the events is computed but not executed (these events are marked as discarded). Let us now ignore the collision type events which are unimportant because they only play a minor role in the total runtime consumption (they are caused by the collisions of points with walls of the bounding area - see Section 3 and by the collisions of the pairs of points). Assuming from this graph, we may state that the redundant computation of topologic events covers nearly as much as 50% of the runtime needed to compute the topologic events. Furthermore, we may assume that there is an upper bound of $O(n)$ and a lower of $O(\log n)$ on both the number of executed and discarded topologic events, which corresponds with the results presented in [Al-

bers et al. 1998], where a naive estimation of the upper bound on the number of the events is given as $O(n^{d+2})$ and its linear improvement is presented.

6.2 Performance in Other Applications

As mentioned in Section 1, we have used our implementation of the described algorithm as a tool for video representation. Utilizing several techniques for stability improvement and overall performance enhancement we were able to successfuly use the kinetic DT as a basic data structure for video representation. A screenshot of the demonstration application is shown in Fig. 9 and additional details on this topic may be found in [Puncman 2008].

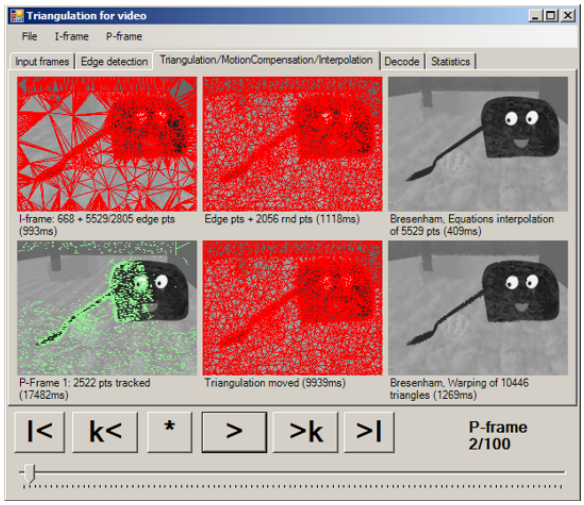

Figure 9: *A screenshots of the video application.*

6.3 2.5D Wave Simulation

For demonstration purposes, we have created a simulation of a 2.5D wave. This simple application consists of a grid of points. Each point in this grid has slightly altered its $y-$coordinate by adding a small random number. One row of these points is then assigned a velocity vector in the form of $[0, y]$. If the moving row collides with the boundary of the triangulation area, it is deflected backwards without any loss of speed. If two points collide, then they switch their velocity vectors, meaning that the previously moving point is now static and vice versa. The points are then assigned a z coordinate by using an equation of a Gaussian curve with its peak being at the coordinates of the moving row of points. This creates the illusion of a wave as shown in Fig. 1.

Animation Download

Some example animations, including the output of the video representation application and the 2.5D wave simulation may be downloaded from:

`http://home.zcu.cz/%7Etvomacka/animations/`

7 Conclusion and Future Work

The presented method for maintaining the kinetic DT uses the Sturm sequences of polynomials combined with the fundamental theorem of algebra in order to estimate the location and multiplicities of the roots of the solved equations and thus is able to discover both the usable times of topologic events and the times which we cannot use. Even although the current version of the application uses only a certain portion of the presented possibilities, we hope that in the near future we will be able to utilize the knowledge in the fields of redundant topological event computation and others in order to vastly improve its performance.

The performed tests show us that even the existing version is usable in certain application, such as the video representation and for demonstration purposes (the simulation of the 2.5D wave). Future improvement capabilities lie in both of these areas and even include extension to higher dimensions, which is possible without any changes in the currently used mathematical apparatus (the described relations may be very easily extended to 3D). On the other hand, capabilities in the field of generalization of the type of movement is nearly impossible because the equations remain usable only for very narrow set of types of movement. This set includes polynomial trajectories but may not be extended to general movement described by nonlinear equations.

Acknowledgements

I would like to thank to my colleague *Petr Puncman* for providing a testing application and many helpful ideas and suggestions, especially in the fields of improving the stability and overall performance of our application.

References

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

DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. 1997. *Computational geometry: algorithms and applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

DEVILLERS, O. 1999. On deletion in delaunay triangulations. In *Symposium on Computational Geometry*, 181–188.

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

GAVRILOVA, M., ROKNE, J., AND GAVRILOV, D. 1996. Dynamic collision detection in computational geometry. In *12th European Workshop on Computational Geometry*, 103–106.

GOLD, C. M., AND CONDAL, A. R. 1995. A spatial data structure integrating GIS and simulation in a marine environment. *Marine Geodesy 18*, 213–228.

HJELLE, O., AND DÆHLEN, M. 2006. *Triangulations and Applications*. Berlin Heidelberg: Springer.

MOSTAFAVI, M. A., GOLD, C., AND DAKOWICZ, M. 2003. Delete and insert operations in Voronoi/Delaunay methods and applications. *Comput. Geosci. 29*, 4, 523–530.

PAN, V. Y. 1997. Solving a polynomial equation: Some history and recent progress. *SIAM Review 39*, 2 (June), 187–220.

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

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

SCHALLER, G., AND MEYER-HERMANN, M. 2004. Kinetic and dynamic delaunay tetrahedralizations in three dimensions. *Computer Physics Communications 162*, 9.

THIBAULT, D., AND GOLD, C. M. 2000. Terrain reconstruction from contours by skeleton construction. *Geoinformatica 4*, 4, 349–373.

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

VOMÁČKA, T. 2008. *Delaunay Triangulation of Moving Points in a Plane, Diploma Thesis*. University of West Bohemia, Pilsen, Czech Republic.

VOMÁČKA, T. 2008. Use of delaunay triangulation of moving points as a data structure for video representation. Tech. rep., University of West Bohemia, Pilsen, Czech Republic.

WEISSTEIN, E. W., 2004. Fundamental theorem of algebra. From MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/QuarticEquation.html.