University of West Bohemia
Faculty of Applied Sciences

Construction of Geometric Models for Moving Points

Ing. Tomáš Vomáčka

Doctoral thesis
submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Computer Science and Engineering

Supervisor: Prof. Dr. Ing. Ivana Kolingerová
Department: Department of Computer Science and Engineering

Pilsen 2019

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Konstrukce geometrických modelů pro pohybující se body

Ing. Tomáš Vomáčka

Disertační práce
k získání akademického titulu doktor
v oboru Informatika a výpočetní technika

Školitel: Prof. Dr. Ing. Ivana Kolingerová
Katedra: Katedra informatiky a výpočetní techniky

Plzeň 2019

Prohlášení

Předkládám tímto k posouzení a obhajobě disertační práci zpracovanou na závěr doktorského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Prohlašuji, že tuto práci jsem zpracoval samostatně s použitím odborné literatury a dostupných pramenů uvedených v seznamu, jenž je součástí této práce.

V Plzni dne 27. srpna 2019

Ing. Tomáš Vomáčka

Contents

Abstract

Kinetic data structures represent a valuable tool for the geometry challenges in computer graphics, because they allow the extension of the standard tools and data structures for time-dependent data. Thanks to this concept, it is possible to exploit the properties of, e.g., Delaunay triangulation for moving entities which can represent for instance pedestrians, airplanes, or entirely abstract data in applications such as video compression or computation of different physical phenomena using the finite elements method. A necessary consequence of the fact, that the primitives ordinarily used to construct these data structures are time-dependent is that the kinetic data structures will change in time. These changes are determined by the so-called kinetic events, which as a matter of fact represent the very basic cornerstone of the discussed topic. Generally, two different types of kinetic events are recognized – external events which directly affect the topology of the data structure and internal events which do not affect the topology but they need to be considered in order to ensure the proper lifecycle of the data structure. In this thesis the focus is aimed at the general analysis of the kinetic events, especially their computation, estimation of their total amount and reduction of the computed and discarded potential events. This analysis is first general and then later applied to concrete examples of kinetic Delaunay triangulation and used to establish the kinetic locally minimal triangulation. Kinetic Delaunay triangulation is also used in two different cases as a tool for potential collision detection in air traffic and as an underlying data structure in a video compression method.

This dissertation thesis was supported by the following projects:

- GA17-07690S: *Methods of Identification and Visualization of Tunnels for Flexible Ligands in Dynamic Proteins*, Czech Science Foundation, 2017–2019.
- SGS-2016-013: *Advanced Graphical and Computing Systems*, University of West Bohemia, 2016–2018.
- SGS-2010-028: *Advanced Computer and Information Systems*, University of West Bohemia, 2013–2015.
- LH11006: *INGEM – Interactive Geometric Models for Simulation of Natural Phenomena and Crowds*, The Ministry of Education, Youth and Sports of the Czech Republic, 2011–2013.
- 201/09/0097: *Triangulated Models for Haptic and Virtual Reality*, Czech Science Foundation, 2009–2011.
- KJB101470701: *Alternative Representation of Image Information Using Triangulations, junior research project*, Czech Science Foundation, 2007–2009.
- LC 06008: *CPG - Center of Computer Graphics - National Network of Fundamental Research Centers*, The Ministry of Education, Youth and Sports of the Czech Republic, 2006–2011.

Abstrakt

Kinetické datové struktury představují cenný nástroj pro řešení geometrických úloh v kontextu počítačové grafiky, neboť umožňují rozšíření standardně používaných nástrojů a datových struktur na data proměnná v čase. Díky tomu je možno využít vlastnosti například Delaunayovy triangulace pro pohyblivé entity, které mohou v kontextu aplikace představovat jako chodce, letadla nebo reprezentovat abstraktní datové entity v aplikacích jako komprese videa nebo výpočet nejrůznějších fyzikálních fenoménů metodou konečných prvků. Vzhledem k tomu, že primitiva, nad nimiž jsou tyto datové struktury běžně vytvářeny, jsou proměnná v čase, musí průběžně docházet také ke změnám v těchto strukturách. Tyto změny jsou určeny tzv. kinetickými událostmi, které zároveň představují základní stavební kámen této problematiky. Obecně jsou rozeznávány dva typy kinetických událostí – vnější, které přímo ovlivňují topologii datové struktury a vnitřní, které topologii neovlivňují, ale je potřeba se jimi zabývat aby bylo zajištěno korektní chování datové struktury v čase. Tato práce se zabývá analýzou kinetických událostí, zejména jejich výpočtem, odhadem jejich počtu a problémem redukce vypočtených, avšak nevyužitých potenciálních událostí. Tato analýza je vedena v obecné rovině a později aplikována na případy kinetické Delaunayovy triangulace a nově odvozené kinetické lokálně minimální triangulace. Konkrétní příklady využití kinetické Delaunayovy triangulace pak představují dvě pokusné aplikace – detekce potenciálních kolizí v kontextu letového provozu a podpůrná datová struktura v aplikaci pro kompresi videa.

Tato disertační práce byla podporována následujícími projekty:

- GA17-07690S: *Metody identifikace a vizualizace tunelů pro flexibilní ligandy v dynamických proteinech*, Grantová agentura České republiky, 2017–2019.
- SGS-2016-013: *Pokročilé grafické a výpočetní systémy*, Západočeská univerzita v Plzni, 2016–2018.
- SGS-2010-028: *Pokročilé počítačové a informační systémy*, Západočeská univerzita v Plzni, 2013–2015.
- LH11006: *Interaktivní geometrické modely pro simulaci přírodních jevů - INGEM*, Ministerstvo školství, mládeže a tělovýchovy České republiky, 2011–2013.
- 201/09/0097: *Triangularizované modely pro haptiku a virtuální realitu*, Grantová agentura České republiky, 2009–2011.
- KJB101470701: *Alternativní reprezentace obrazové informace s využitím triangulací*, Akademie věd České republiky, 2007–2009.
- LC 06008: *Centrum počítačové grafiky*, Ministerstvo školství, mládeže a tělovýchovy České republiky, 2006–2011.

Acknowledgement

I dedicate this thesis to my late father, who has brought me to the academy career and always served as an endless source of inspiration and support.

I would like to thank my thesis advisor Ivana Kolingerová for great support from the beginning to the end. And to all the people who helped me along the way with research, advice and consultation; Martin Maňák, Světlana Tomiczková, Petr Puncman, to name just a few. Last but not least, many thanks belong to my family for always believing I will finish my thesis eventually.

Chapter 1

Introduction

Traditionally, the field of computer graphics include a wide variety of geometry-based challenges, such as geometric sorting, path planning, exploring different relationships within data sets, and many others. In order to solve these problems, various algorithms and data structures are employed, specific to the task at hand. Since the challenges are very often tied to specific real-life scenarios, it is natural to expect that the environment will change with the passage of time and this may result in variations of the data set we are using to tackle the problem – for instance one might be trying to plan a path of an autonomous agent in an environment with moving obstacles. These agents may represent pedestrians, groups of animals of different kinds, or autonomous robots. A variation of this task would be an early warning system that detects possible collision trajectories among planes flying in a certain area.

In order to be able to handle a set of moving data, the commonly used spatial division data structures have to be modified so that they become able to incorporate the movement. There are several ways of doing so – the most straighforward approach is to discretize the movement and exploit such a set of tools that allow both addition and removal of the constructing primitives into and from the data structure. Movement is then simulated by removing and reinserting the primitives on new positions according to their trajectories. Such structures are then called *dynamic*. The other commonly used approach is based on the geometric features specific to the data structures. By computing the times when the data structure reaches a singular state due to the movement of the underlying data, it is possible to maintain its properties by introducing a certain kind of updates. Structures of this kind are then called *kinetic*. This work focuses almost exclusively on the kinetic data structures.

Similarly to the ordinary spatial division structures, their kinetic counterparts divide the given space (usually the Euclidean plane) into cells corresponding to the individual generators, according to a certain set of rules. The generators are in our case given as a set of points, but sometimes it is convenient to use more sophisticated generators, such as weighted points, line segments, circles, general polygons or even more complex primitives, depending on the intended application. The construction rules determine the type and properties of the spatial division and most commonly take the form of evaluating an algebraic function.

The most commonly used kinetic data structures are represented by kinetic Voronoi diagram and especially by its dual structure – kinetic Delaunay triangulation – and their modifications [45, 64]. The cells produced by using ordinary Voronoi diagram are composed of points that are closer to their generator (each cell belongs to one of the generators) than to any other primitive in the generator set. Together with the motion of the points inside the generator set, this basic feature is

most commonly used in applications where general location or proximity of the generators plays an important role, such as collision detection as in [31], navigation in the virtual environments shown in [14, 34], mesh generation for various purposes, such as fininte element method [11] and many others. The Delaunay triangulation is a structure dual to Voronoi diagram and thus has exactly the same features. Moreover, the triangles produced by the Delaunay triangulation are usually of very good quality (close to equilateral, prolonged and narrow triangles occur rarely) which makes it very useful for various applications where a triangular mesh needs to be generated and its quality is important.

Since the introduction of kinetic data structures in [6], many scientists have focused not only on the utilization of this powerful tool in various environment, but also on the general analysis of its behavior, with the most important questions being asked about the nature and amounts of the changes occuring during their lifecycle. Since one cannot employ the usual tools for data structure analysis due to the ever-changing nature of the structure itself, the analysis of kinetic data structures differs significantly and a completely new way of thinking has to be introduced [8].

1.1 Problem Definition

The aim of this thesis is to explore the process of kinetization of the planar spatial data structures with special respect to the event computation and role in the data structure lifecycle. The first objective is to analyze the event computation process in kinetic data structures in order to determine which of the events may be omitted from execution before or after they are computed. The acquired knowledge is then used to propose the application of kinetization principle on a locally minimal triangulation and the obtained theoretical results are compared to the well-known kinetic Delaunay triangulation. The last objective of this thesis is to evaluate the obtained theoretical results in the context of various applications.

1.2 Summary of Contributions

The first three contributions [92,94,95] deal with kinetic Delaunay triangulation principles with a special focus on event computation methods and event redundancy research. The principle of the introduced improvements lies in using a Sturm sequence of polynomials to determine intervals of time which contain the roots that determine the time of event occurence. Combined with knowledge about the data structure behavior, this allows a non-negligible portion of the events to be discarded before they are even computed and therefore improves the performance and stability of the data structure.

The fourth contribution [96] introduces a novel video compression method based on kinetic Delaunay triangulation which was created in cooperation with *Petr Puncman* from the University of West Bohemia. The triangulation is constructed over a collection of samples from selected intra-coded frames, point movement vectors are determined by precomputed optical flow vectors gained by block matching algorithm in inter-coded frames and decoding data from samples by means of barycentric interpolation and feature-based warping.

The fifth contribution is an application of kinetic Delaunay triangulation for early warning system in air traffic control [90]. The application is an extension of a collision detection system with some features specific to the air traffic – since we consider a real life use case scenario,

the movement of the points in the triangulation is very specific as the updates on the airplane positions come only when they are acquired by the radar and therefore the triangulation needs to be periodically updated because the planes may be too far away from the predicted positions from the previously available data.

The next group of contributions [53,54,91] focuses on the features of kinetic locally minimal triangulation with respect to the general properties of this data structure and comparison to kinetic Delaunay triangulation which might be the best known type of kinetic triangulation to this date. Using advanced statistical methods, the bounds on the number of events in these types of triangulations are estimated and compared.

Finally, the last contribution is represented by a single unpublished manuscript created in cooperation with *Jakub Szkandera*, a researcher from the University of West Bohemia. It discusses the problem of large scale pedestrian navigation in an environment with varying preference levels. The manuscript is currently available online, see [97].

1.3 Organization of the Thesis

The first part of this thesis summarizes the basics of spatial data structures that are often used in computational geometry. Chapter 2 discusses the basic principles of selected data structures and outlines the basic rules that must be adhered to in order to validate these structures. Chapter 3 discusses some of the construction algorithms and their properties which may become important when related to the kinetic data structures. The kinetic data structures themselves are introduced in Chapter 4 and some of the examples of their use in different fields of expertise are provided in Chapter 5.

The focus of the second part of this thesis is on the process of kinetization and the statistical evaluation of the kinetic data structures. Chapter 6 describes the lifecycle, the events and the general properties of kinetic Delaunay triangulation and kinetic locally minimal triangulation. Chapter 7 provides an overview of the most commonly used methods for event time computation and delves into the topic of event redundancy reduction and specialized computation methods that may be exploited in order to improve the performance and stability in kinetic applications.

The third and last part of this thesis summarizes the contributions and applications. Chapter 8 describes how the different methods described in the earlier chapter may be utilized in order to create an enhanced implementation of kinetic Delaunay triangulation where the main focus is on event redundancy reduction. Chapter 9 shows a novel video compression method based on kinetic Delaunay triangulation. Chapter 10 describes an application of kinetic Delaunay triangulation as a tool for early warning system in air traffic control. Chapter 11 provides an overview of a basic research in the related field of path planning in the context of crowd simulation. The thesis is concluded in Chapter 12.

Part I

State of the Art and Examples

Chapter 2

Spatial Data Structures

In order to handle the relations within a set of moving data one has to deal with many different data structures, purpose of which is to subdivide the $2-$dimensional space into regions according to a specific set of rules – spatial division data structures. This chapter describes some of their most commonly used types from the static point of view; the exact way of handling the movement of the underlying points in the generator set will be shown later in Chapter 6.

The discussed data structures are usually defined by functions of the input data (e.g., the coordinates of the given points). These functions are called *predicates*. Some of the common predicates used in computational geometry will be described in this chapter and will be referred to in Chapter 4.

2.1 Voronoi Diagram

Let $S = \{p_1, \ldots, p_n\}$ be a set of n points (generators) in $\mathbb{R}^2$, then let us define:

Definition 1 (Voronoi cell)**.** *A set of points $V(p_i)$, where:*

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

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

Definition 2 (Voronoi diagram)**.** *A set of Voronoi cells*

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

is called a Voronoi diagram of S.

The boundaries of the Voronoi cells are also called the *Voronoi edges* and are composed of such points that are equally distant to two of the generators. Points of conjuction of the Voronoi edges are called *Voronoi vertices* and represent points that are equally close to three or more of the generators.

2.2 Delaunay Triangulation

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

Definition 3 (Planar triangulation, [45, 98])**.** *A set of edges in $\mathbb{R}^2$ that fulfill the following conditions:*

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

Delaunay triangulation may now be defined as a special case of a planar triangulation:

Definition 4 (Delaunay triangulation, [45])**.** *Triangulation $T(S)$ that fulfills the condition:*

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

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

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

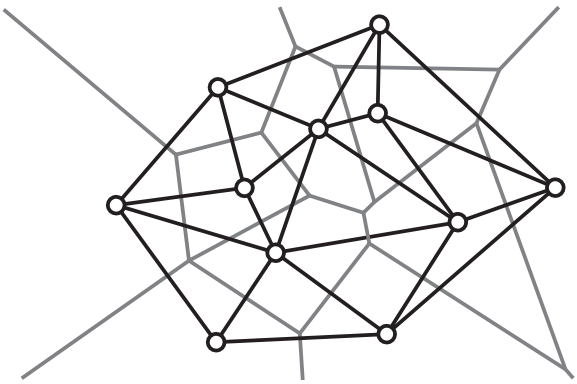


Figure 2.1: Voronoi diagram (grey lines) and its dual structure, the Delaunay triangulation (black lines).

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

2.2.1 Properties of Delaunay Triangulation

Uniqueness and Singular Configurations

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

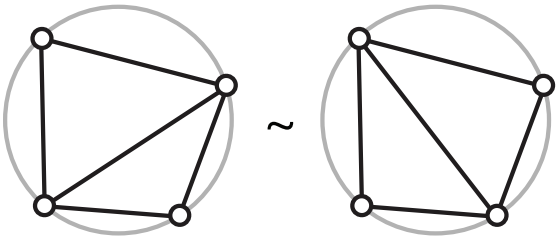


Figure 2.2: Two legal topologic configurations in Delaunay triangulation.

Locality

As shown by Delaunay himself in [19], any triangulation which is locally Delaunay, is also globally Delaunay. This means that if each pair of adjacent triangles in the triangulation fulfills the abovementioned Delaunay condition (i.e., if we construct the circumcircle of arbitrary one of the two adjacent triangles, the remaining point of the quadruple will not lie inside this circle), then each point in the triangulation fullfills this condition against each of its triangles.

Higher Dimension Embedding

Let us first define some of the terms we will need in order to inspect the relationship between Delaunay triangulations in $\mathbb{R}^2$ and convex hulls in $\mathbb{R}^3$:

Definition 5 (Lifted point, lifted space, [24])**.** *Given a point $p = (x,y)$ in $\mathbb{R}^2$, its lifted point $p^+ \in \mathbb{R}^3$ is given as:*

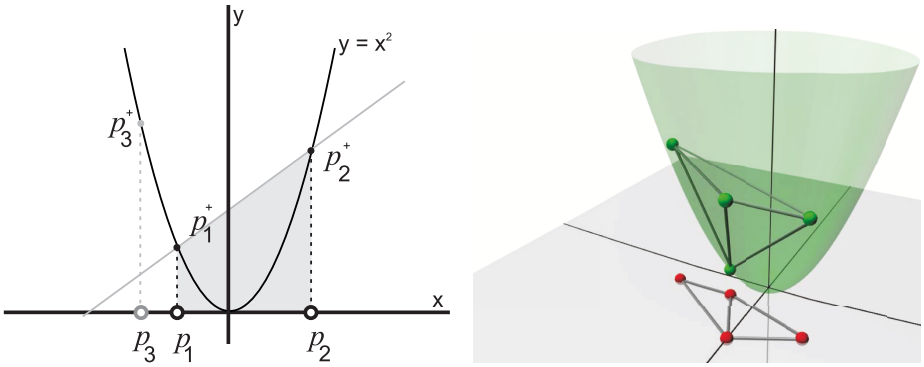
$$p^+ = (x,y,x^2+y^2) \tag{2.3}$$

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

It is well known from [24, 45] that Delaunay triangulations (as well as Voronoi diagrams) are closely related to higher dimension convex hulls. Given a set of points S and a set of corresponding lifted points $S^+ = \{p_1^+, p_2^+, \ldots, p_n^+\}$, the Delaunay triangulation $DT(S)$ will be equal to planar projection of lower $CH(S^+)$.

In other words, if the points in the generator set are projected on a paraboloid $z = x^2 + y^2$, a convex hull of this projection is constructed and the lower facets of this convex hull are projected

back onto the original plane using a planar projection, a Delaunay triangulation of the original point set is obtained. An illustration of this relationship is given in Fig. 2.3.



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

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

A set of points in $\mathbb{R}^2$ may be seen in this figure, represented by the red balls. These points are then projected on a paraboloid using the relation in Eq. (2.3) – the projections are marked by the green balls. A convex hull constructed over the set of the projections is shown and its lower facets are then projected back into $\mathbb{R}^2$ where they form the Delaunay triangulation of the original (red) points.

Nearest Neighbors

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

2.2.2 Incircle Test

The Delaunay criterion requires that no point $p_i \in S$ in the generator set lies inside a circumcircle of any of the triangles in $DT(S)$. There are numerous ways to determine whether a point lies inside a circumcircle of a given triangle, but the most commonly used one is based on the relationship between the Delaunay triangulation and the higher dimension convex hull which was described earlier. An illustration is shown in Fig. 2.3(a), where a one-dimensional illustration of the situation can be seen – the lifted points p_i^+ are projected onto a parabola. In order to determine if p_3 lies between p_1 and p_2 one has to check if p_3^+ lies below the line defined by p_1^+ and p_2^+. This one-dimensional variant of this test is in fact the well known orientation test. The situation in higher dimensions is then analogical.

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

$$incircle(p_i, p_j, p_k, p_l) = \det \begin{bmatrix} x_i & y_i & x_i^2 + y_i^2 & 1 \\ x_j & y_j & x_j^2 + y_j^2 & 1 \\ x_k & y_k & x_k^2 + y_k^2 & 1 \\ x_l & y_l & x_l^2 + y_l^2 & 1 \end{bmatrix} \tag{2.4}$$

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

If the triangle defined by points $p_i p_j p_k$ is oriented counterclockwise, a positive value of Eq. (2.4) means that p_l lies inside the circumcircle of $p_i p_j p_k$, a negative value means that p_l lies outside and a zero value shows that the point lies exactly on the circumcircle. If the orientation of the points $p_i p_j p_k$ is unknown, an orientation test needs to be made together with the incircle test [45]:

$$orient(p_i, p_j, p_k) = \det \begin{bmatrix} x_i & y_i & 1 \\ x_j & y_j & 1 \\ x_k & y_k & 1 \end{bmatrix} \tag{2.5}$$

The $incircle(p_i, p_j, p_k, p_l)$ value may be then computed as follows:

$$incircle(p_i, p_j, p_k, p_l) = incircle'(p_i, p_j, p_k, p_l) \cdot orient(p_i, p_j, p_k) \tag{2.6}$$

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

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

2.3 Locally Minimal Triangulation

Another special case of planar triangulation described in this chapter is the locally minimal triangulation (*LMT*):

Definition 6 (Locally minimal triangulation, [75])**.** *A triangulation of a set of points* $S = \{p_1, p_2, \ldots, p_n\}, n > 2, S \subset \mathbb{R}^2$, *is locally minimal if and only if every edge* $p_i p_j$ *shared by two triangles* $p_i p_j p_k$, $p_i p_j p_l$ *forming a convex quadrilateral is not longer than the diagonal* $p_k p_l$.

2.3.1 Properties of Locally Minimal Triangulation

Local Minimality Condition

As given in Definition 6, $LMT(S)$ is proven valid by comparing the lengths of the two possible diagonals in convex quadrilaterals, this gives us two different conditions that need to be

checked - the length comparison itself and the convexity check of the four points. Let $S_Q = (p_i, p_j, p_k, p_l), S_Q \subseteq S$ be a subset containing four points which create two adjacent triangles $p_i p_j p_k$ and $p_i p_j p_l$. Every such quadrilateral in $LMT(S)$ will be checked thus creating two predicatess for the local minimum $LM(S_Q)$ and convexity $CH(S_Q)$:

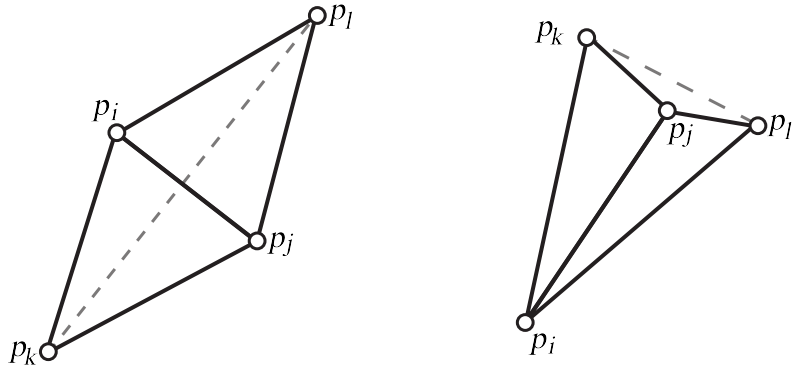
$$LM(S_Q) = |p_i - p_j| - |p_l - p_k| \tag{2.7}$$

$$CH(S_Q) = orient(p_k, p_l, p_i) \cdot orient(p_k, p_l, p_j) \tag{2.8}$$

Eq. (2.7) compares the lengths of the two possible diagonals inside the quadrilateral S_Q and from practical point of view, it makes perfect sense to simplify it by comparing square length, therefore simplifying the computation by removing one square root operation per length compared. Introducing the second predicate in Eq. (2.8) is necessary because the check for a shorter diagonal in any given quadrilateral is only valid if the convexity requirement is met. As we may see, this predicate is composed of two orientation tests and determines if both of the vertices connected by the inner diagonal are on the same side of the line segment defined by the other two vertices.

Uniqueness and Convex Subsets

Unlike the Delaunay triangulation that was mentioned earlier, $LMT(S)$ is not unique for S and it is dependent on the concrete method of its construction. Also note the fact that the local minimality is only ensured for *convex* qaudrilaterals – if a quadrilateral formed by two adjacent triangles in *LMT(S)* is not convex, it is generally not possible to make sure that the shorter of its diagonals will be present as an edge in the triangulation, see Fig. 2.4 which shows two basic examples. As you can see, the convex quadrilateral in Fig. 2.4(a) is formed by the two triangles sharing its shorter diagonal. On the other hand, the diagonals in the quadrilateral shown in Fig. 2.4(b) cannot be swapped because it is not convex.



(a) Convex quadrilateral. (b) Nonconvex quadrilateral.

Figure 2.4: Possible quadrilateral configurations.

Chapter 3

Construction Algorithms

Many different construction algorithms are commonly used in order to create various types of spatial division data structures. This chapter provides a brief overview of some selected examples of the construction methods together with some of the data structures mentioned in the previous chapter. Furthermore, a more detailed description of the incremental insertion algorithm is given with a special regard to the construction of the Delaunay triangulation, because its properties make it extremely suitable for later use in its kinetization.

3.1 Construction Method Properties

There are numerous properties that may be used to compare construction algorithms, [52]. Among them, the following provide valuable information in the context of kinetic data structures (for others, see [16]):

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

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

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

Extensibility to higher dimensions: even though this thesis focuses on planar data exclusively and thus higher dimension functionality is of rather low importance, some of the described algorithms may not be extended to higher dimensions which will handicap them if this kind of a future development is considered.

3.2 Construction Methods Overview

The general complexity of planar Delaunay triangulation construction is $O(n \log n)$ in the worst case as shown in [16], but the exact time complexity depends on the algorithm used; there are algorithms that run in $O(n)$ expected time. The following list provides a brief overview of some of the most commonly used algorithms, [52].

Higher Dimension Embedding: as shown before, there is a relation between $n-$dimensional Delaunay triangulations and $n+1-$dimensional convex hulls which may be used for the purposes of DT construction [7]. It is obvious that the properties of this algorithm depend solely on the used method for convex hull construction. The time complexity is in the optimal case equal to $O(n \log n)$.

Divide and Conquer: this type of algorithms is based on the idea that the initial dataset may be divided recursively – partial *sub*-triangulations are then created for the divided parts and connected together [16]. Divide and conquer algorithms are usually time-optimal in the worst case, even though they are usually untrivial to implement. This approach is not online, is easy to parallel but the extensibility to higher dimensions is difficult because of problems with sorting in higher dimensions. The time complexity of this approach is $O(n \log n)$ in the worst case.

Local Optimization: starting with a random initial triangulation of the given dataset, the local optimality of Delaunay triangulation may be exploited. By testing the satisfaction of the Delaunay property for each pair of adjacent triangles, the locations where the Delaunay property is violated are discovered and the given triangulation is converted to DT by swapping the common edge of these triangle pairs. This method is not online, may not be reliably extended to three dimensions, is difficult to parallelize and its performance is dependent on the algorithm used for creating the initial construction. The time complexity of this approach is determined by the number of performed edge swaps. In $\mathbb{R}^2$, it is $O(n^2)$ in the worst case and $O(n)$ in the expected case. Details may be found in [45, 49].

Incremental Construction: Delaunay triangulation may be constructed by starting with the shortest edge from the given dataset and successive construction of such triangles, that fulfill the Delaunay condition [20]. The time complexity of this approach is $O(n^3)$ in two dimensions if we do not use any acceleration techniques. Using data structures as directed acyclic graphs [18] or advanced point location algorithms such as different types of walks [23], the expected time complexity of $O(n \log n)$ can be achieved.

Sweep Construction: after sorting the given points along an axis, the Delaunay triangulation is created by adding them to a partial triangulation in the order given by the sorting. This approach is not online, may be parallelized, is extensible to three dimensions and runs in $O(n \log n)$ time. Examples of this approach may be found in [29, 79]. Even though the properties of this algorithm make it slightly inconvenient for the use in kinetic data structures management, it plays very significant role in the general principle of kinetization, see Chapter 4 for further details.

Incremental Insertion: starting with an initial triangle large enough to contain the given dataset, the points from the dataset are added, one at a time. The triangle that contains the added point is divided and the triangulation is locally repaired if the Delaunay condition is broken

in the process. This algorithm is online, may be paralellized and extended to three dimensions. Its time and memory complexity is derived from the used point location method. It will be described in detail in the following section.

3.3 Incremental Insertion Algorithm Details

3.3.1 Overall Functionality

The functionality of the incremental insertion algorithm [18] for constructing a Delaunay triangulation is shown in Alg. 1. The algorithm consists of four distinct parts – first, the initial simplex is constructed that contains the whole area covered by the points in S, after that, the points from the generator set are separately added to the current triangulation (which contains only the auxiliary simplex at the beginning of this step) and during the addition of the points, the triangulation is being constantly checked for edges that do not satisfy the Delaunay condition and these edges are replaced by the other possible edges by swapping. This process is called edge legalization. Finally, all the edges connecting the points from the auxiliary simplex with any other point are removed from the triangulation (this step is usually not performed in the kinetic applications, see further).

Algorithm 1: Overall functionality of the incremental insertion algorithm for DT construction.

Input: Set $S = \{p_1, p_2, \ldots, p_n\}$ of n distinct points
Output: $DT(S)$

1 $DT(S) = p_0 p_{-1} p_{-2}$ containing S `// Initialization`
2
3 **foreach** $p_r \in S$ **do**
4 Localize the triangle $p_i p_j p_k$ such that $p_r \in p_i p_j p_k$
5 **if** p_r *is inside* $p_i p_j p_k$ **then**
6 $DT(S) = DT(S) \setminus \{p_i p_j p_k\}$
7 $DT(S) = DT(S) + \{p_r p_i p_j, p_r p_j p_k, p_r p_k p_i\}$
`// ` $p_i p_j p_k$ `is split into three triangles`
8 **else** `// ` p_r `lies on the edge common to` $p_i p_j p_k$ `and` $p_i p_j p_l$
`// The two adjacent triangles are split into four`
9 $DT(S) = DT(S) \setminus \{p_i p_j p_k, p_i p_j p_l\}$
10 $DT(S) = DT(S) + \{p_r p_i p_k, p_r p_j p_k, p_r p_l p_i, p_r p_l p_j\}$
11 **end**
12 Legalize all newly created edges `// Details in Alg. 2`
13 **end**
14 Remove all the edges containing $p_q \in \{p_0, p_{-1}, p_{-2}\}$ if needed

3.3.2 Initial Triangle Construction

In order to be able to locate the point location performed in the following step, it is necessary to make sure that a triangle containing the added point exists. This can be done by encapsulating the whole area covered by the triangulation by a single simplex (i.e., a triangle $\mathbb{R}^2$, a tetrahedron

in $\mathbb{R}^3$, etc.) that is large enough not to alter the edges of the convex hull of the original set. On the other hand, it must not be too large, because otherwise it could make the construction of the triangulation numerically unstable. The correct size of these simplices has been adressed by [48, 98] and experiments show that the ideal way to construct the initial simplex is the one illustrated in Fig. 3.1.

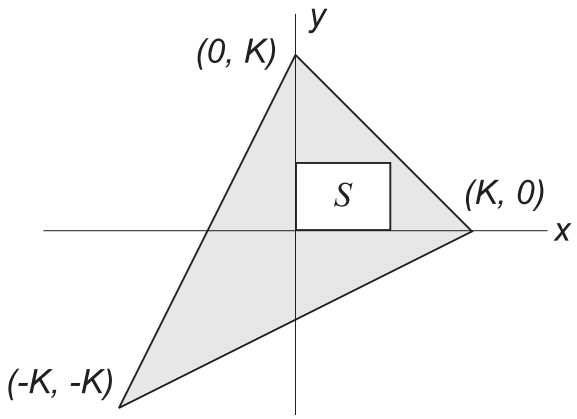


Figure 3.1: The ideal initial simplex construction for the incremental insertion algorithm, [49].

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

3.3.3 Point Location

As shown in Alg. 1, in order to add a point into the triangulation, the triangle containing its location has to be found. Generally, there are two different classes of approaches for point location – the approaches based on the walking algorithms and the approaches based on special location data structures.

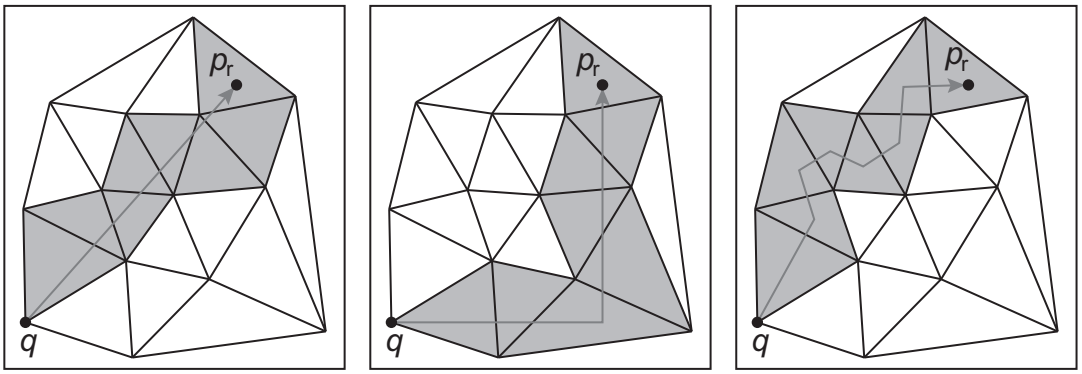
The walking algorithms take advantage of the fact that no special data structure is needed. Each point location process starts in an arbitrary triangle and by following a given set of conditions traverses the triangulation until the triangle containing the given point p_r is reached. According to [23, 81], there are three main types of the walking algorithms, based on the type of traversal they use – see Fig. 3.2 (all the walks start in the triangle containing the point q).

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

Orthogonal walk: Two axis-aligned line segments are created that connect the starting triangle with the given point p_r. The triangulation is then traversed along these two line segments. According to [81], the traversal may not end in the correct triangle and it is necessary to combine it with some other traversal type.

Stochastic walk: The next triangle of the stochastic walk algorithms traversal is chosen by randomly picking an edge of the currently processed triangle and testing if the target point

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



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

Figure 3.2: Comparison of different walking strategies, [80].

As shown in [61, 81] and others, the time needed by the walking algorithms to locate a point strongly depends on the selection of the starting triangle. Depending on the choice of the starting triangle, the expected time complexity of a walk algorithm may vary from $O(n^{1/2})$ to $O(n^{1/4})$ which is worse than the optimal $O(\log n)$.

The other approach for triangle location uses special auxiliary data structures such as Directed Acyclic Graphs (*DAG*), skip-lists and others [18, 22, 48]. These special data structures usually work by creating a hierarchy of the triangles in the triangulation. Let us have a look at the *DAG* structure – as can be seen in Fig. 3.3, where a very simple version of *DAG* is shown. In this figure, a simple triangulation is shown as the initial state. The associated *DAG* structure starts with a pointer to each of its triangles. As new points are being added into the triangulation and the edges are being swapped in order to maintain the Delaunay property of the triangulation, we may see that two things happen in the *DAG*. When a triangle is subdivided into three triangles ($T_2 \rightarrow \{T_4, T_5, T_6\}$) a new level in the *DAG* is created and the original triangle is connected to its children with pointers. When an edge swap occurs, all the upper level triangles that contain at least part of the newly created triangles are connected to them via new pointers. As we can easily see, the *DAG* structure is organized in a tree-like fashion and thus enables point location with expected time complexity of $O(\log n)$.

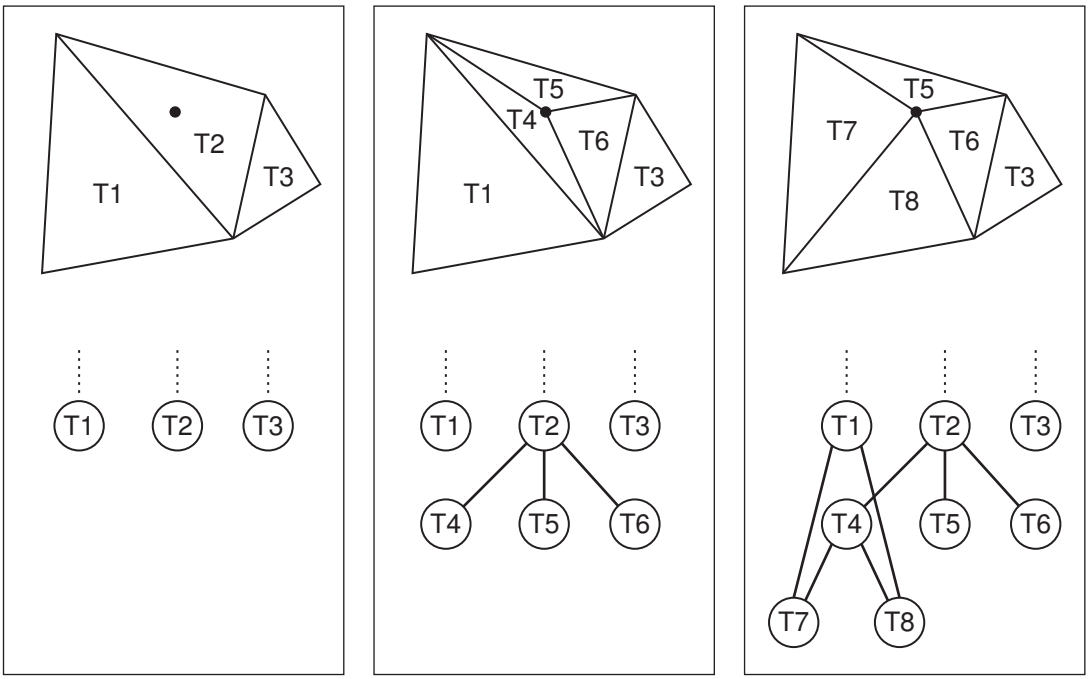
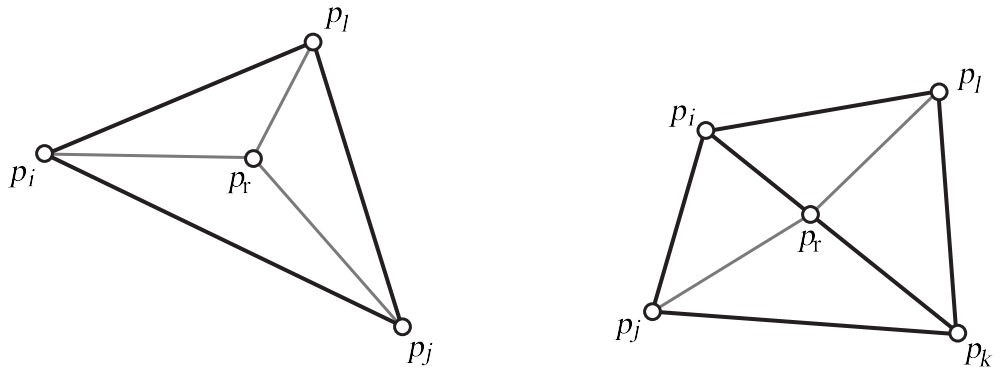


Figure 3.3: A simple triangulation and an associated *DAG*, [80].

3.3.4 Point Insertion and Edge Legalization

Once the triangle $p_i p_j p_k$ containing the given point p_r is located, two cases may occur (considering two dimensional triangulation) – p_r may either lie inside $p_i p_j p_k$ or on one of its edges. Theoretically, a third special case might occur when p_r is identical to one of the vertices of the target triangle, but only if we allow the set of input data S contain multiple identical points (as shown in Alg. 1, we require S to contain *n distinct* points). The process of handling both the allowed cases is similar – see Fig. 3.4.



(a) p_r is added inside a triangle.

(b) p_r is added on an existing edge.

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

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

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

Definition 7 (Edge swap)**.** *Given two adjacent triangles $p_ip_jp_k$ and $p_jp_kp_l$, an edge swap performed on these triangles replaces them with triangles $p_ip_jp_l$ and $p_ip_lp_k$.*

After the illegal edges are swapped, the legality test is recursively performed on the outer edges of the new triangles, created by the swapping. This process is shown in Alg. 2.

Algorithm 2: Edge legalization in the process of $DT(S)$ construction.

Input:

- p_r – a point being inserted into $T(S)$
- p_ip_j – the edge of $T(S)$ that may need to be swapped
- $T(S)$ – a triangulation

Output: $DT(S)$ – Delaunay triangulation

```
1 if p_i p_j is illegal then
2     Let p_i p_j p_k be the triangle adjacent to p_r p_i p_j along p_i p_j
3     Replace p_i p_j with p_r p_k                          // Swap p_i p_j
4     LegalizeEdge(p_r, p_i p_k, T(S))
5     LegalizeEdge(p_r, p_k p_j, T(S))
6 end
7 Return DT(S)                                              // T(S) ≡ DT(S)
```

Chapter 4

Kinetic Data Structures

This chapter describes the general principles of kinetic data structures. The general process of static structure kinetization is discussed as well as the most commonly used methods of evaluation of the kinetic data structure properties. First, the difference between the dynamic and the kinetic approach to point movement is described. After that, general functionality of the kinetic data structures is discussed in general. The last part of this chapter provides the tools for statistical analysis of kinetic data structures. This knowledge and tools will be used to deduce the properties of concrete data structures in later chapters.

4.1 Kinetic Data Structure

Kinetic data structure (*KDS*) is defined informally in [8] as a pair of two structures: a proof of correctnes of an attribute and a priority queue. This essentially means that such a data structure could be constructed over a set of time-dependent generators and is valid with respect to the given set of certificates for any value of time.

4.2 Kinetic & Dynamic Approach

In general, there are two different approaches for handling movement in the given dataset (in the case of this thesis, it is a set of planar points). The movement may be discretized and simulated by removing the points from the given data structure and reinserting them back at new positions as if they moved there or it may be understood as a continuous change of the dataset. To distinguish between these two approaches, the discrete one is often referred to as the dynamic movement whereas the continuous one is often considered to be the true kinetic approach [9].

Dynamic approach: there are several advantages of the dynamic approach – first of all, its performance is completely independent of the type of the movement of the points. The points may move along any type of trajectory with no restrictions on acceleration or other movement properties. The runtime consumption is then dependent on the algorithms for point insertion and removal. On the other hand, the discretization of movement generates some drawbacks, for instance it cannot be reliably used for collision detection and similar problems; [59] shows that this type of data manipulation is most commonly used for simulating

time-dependent datasets where some points are only used for a limited time period. Generally speaking, we may say that the dynamic approach to movement simulation tends to suffer from two problems – either it is oversampled and we are wasting computational resources on computing unnecessary data, or it is undersampled and we miss important events.

Kinetic approach: the over/under-sampling difficulties of the dynamic approach may be overcome by using the kinetic approach – by computing a certain time events we are able to determine when the underlying data structure changes and needs our attention (topologic update, etc.). This type of simulation is then obviously strongly dependent on the type of the underlying data structure and the type of movement as these two properties determine the type of computation that needs to be performed in order to keep its properties. However, the data can be accessed at any given time without a loss of information. Furthermore, according to [36,74], in certain applications it is faster to simulate the movement using this type of approach. Very similar principles can also be used even in the case that the triangulation does not move in the usual sense, but is only *perturbated* and needs to be quicky updated according to the new positions of generator as in [104].

4.3 Kinetic Data Structure Cornerstones

In order to be able to kinetize an ordinary (i.e., static) data structure, one has to understand the basic cornerstones of the kinetization process. These will be described in this section. The described properties are general and may be applied to any type of kinetic data structure.

4.3.1 Predicates and Certificates

As shown in Chapter 2, the spatial division data structures are often defined by special functions called predicates. Examples of such predicates include the incircle and orientation test as shown in Eq. (2.4) and Eq. (2.5) respectively. As shown in [9,36], each geometric structure constructed over a set of geometric primitives may be proven valid by checking a finite numbers of predicates of these primitives. These checks are then called *certificates*. In the case of Delaunay triangulation, the certificates are represented by the incircle test functions Eq. (2.6) as we are able to determine if any given triangulation $T(S)$ is Delaunay by performing the incicrcle test on each pair of adjacent triangles in the triangulation (if S is a finite set, then the triangulation will have a finite number of edges and thus we will have to perform a finite number of tests).

4.3.2 Point Movement Description

A moving point is such a point which has time-dependent coordinates:

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

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

Theoretically, it is possible to use other functions than polynomials to describe the point movement, but only polynomials are practically used. This is caused by the fact that they provide us with a relatively wide variety of options of the point behavior and their roots are relatively easy to compute. The polynomials also represent one of the best choices because we usually require the given functions to behave in an algebraic (or at least in *pseudo*-algebraic) fashion, which means that we require the functions to have a finite number of roots [37]. Furthermore, the polynomials (albeit of high degrees) may be used to interpolate almost any physically tangible trajectory, for instance by using the Taylor expansion [2].

Note: In order to eveluate one of the basic properties of kinetic data structures described later (namely the strong efficiency), any type of movement of the underlying point set needs to be considered. This will be the only exception to the polynomial movement allowed in the following text.

4.3.3 Certificate Functions

If the points are allowed to move continuously by replacing their coordinates by functions of time, the certificates themselves will become functions of time – *certificate functions*. The type of these functions depends both on the original certificates and on the type of point movement we allow. As we have already mentioned in Chapter 2, the certificates are functions of the input data.

4.3.4 Kinetic Events

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

On the other hand, in common situations, instants in time will usually occur when the sign of the certificate function changes. These instants indicate that the structure is about to lose its properties and in order to preserve its validity, a change in the structure needs to be performed (such as the case of a point entering the circumcircle in Delaunay triangulation). These instants are called *kinetic events* (or *events* for short) because they are caused by the movement of the points, or certificate failure times because they denote that the certificate function has failed.

Internal and External Kinetic Events

Two types of kinetic events are usually described in the literature [9, 37] – the *internal* and the *external* kinetic events. The difference between these two types of events is that the external events as defined in [37] are those which directly change the topology of the kinetic data structure. An example of an external event is the edge swap in the kinetic Delaunay triangulation – it needs to be done in order to retain the Delaunay property of the triangulation and it changes its topology. The internal events only exist in some kinetic data structures (e.g., in the kinetic locally minimal

triangulation) but cannot be found in others, such as the kinetic Delaunay triangulations. These events are not "visible from the outside", they are such events that need to be processed for the *KDS* to work correctly but do not directly change its topology.

Event Queue

In order to be able to manage the kinetic structure continuously, it is vital to process the kinetic events in the correct order. If two different certificates fail (i.e., the corresponding certificate functions change signs), it is necessary to process the changes in the kinetic data structures in the correct order. A priority queue is most commonly used for this task [9] – the certificate failures are stored in the queue ordered by the time when they occur.

Lifecycle of Kinetic Data Structures

The management of a kinetic data structure generally consists of two phases [5, 8, 31] – the *initialization* phase, where the data structure itself is constructed and the *update* phase, which is periodically repeated as long as needed. The lifecycle is summarized by Algorithm 3.

The initialization phase of Alg. 3 consists of two steps – first of all, the desired data structure is constructed using an arbitrary construction algorithm as described in Chapter 3 and once it is created, the set of nearest kinetic events is computed. The exact process of the initial events computation is dependent on the concrete data structure being used. For example in the case of kinetic Delaunay triangulation, for each pair of adjacent triangles such a time is computed, when their four points become cocircular.

All the initial events are then placed into a priority queue with the priority defined in such a way to ensure 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:

$$p = -t_{event} \tag{4.2}$$

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.

The update phase of the algorithm consists of locally repeating the iteration step each time the value of time has changed. This step consists solely of popping the events from the top of the priority queue, executing them (thus possibly changing the topology of the data structure or some of its features) and pushing new events into the queue, if any. 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 value of current time. The current value of time is then increased as necessary for the given application.

4.4 Kinetic Data Structures General Properties

According to [9,10,37], there are four basic properties that may be used to judge the effectiveness of a particular data structure for kinetic data. These are *responsiveness, efficiency, locality and compactness*. Note that these properties are dependent on the exact implementation of the given kinetic data structure, not only on its type and the property it is maintaining. For the purposes of

Algorithm 3: General lifecycle of kinetic data structures.

Input:

- Q – Priority queue
- $S = \{p_1, p_2, \ldots, p_n\}$ – set of time-dependent points

Output:

- Continually legalized kinetic data structure

Auxiliary:

- t – The current time of the triangulation
- E – temporary event variable

```
   // Initialization phase
1
2  Create KDS(S) using arbitrary construction algorithm
3  t ← 0                                   // Set current value of time to 0
4  foreach Primitive n−tuple in KDS(S) do
5  |  Compute the next future event E_i at time t_i
6  |  if E_i exists then
7  |  |  Q.push(E_i, t_i)
8  |  end
9  end
   // Update phase
10 while t < t_max do
11 |  while Time of Q.head() > t do
12 |  |  E ← Q.pop()
13 |  |  Execute E
14 |  |  Push new events into Q as required
15 |  end
16 |  t ← t + Δt
17 end
```

the *KDS* properties description, let us denote that a given *KDS* contains n moving primitives. Let us also denote that a function is small if it is bounded by $O(\log^{\varepsilon}(n))$ or $O(n^{\varepsilon})$, for an arbitrarily small $\varepsilon > 0$.

Responsiveness: A responsive kinetic data structure is such a *KDS* that takes only a small amount of time to handle every certificate failure (an internal or external event). Handling of the event may include changing the topology of the data structure, altering its properties, adding new certificate functions or removing old ones, etc. If expressed as a function of n, a *KDS* is responsive if the time required to handle every certificate failure is small.

(Weak) Efficiency: *KDS* is said to be efficient if the total number of events processed in the worst case is asymptotically the same as (or slightly larger than) the number of external

events processed in the worst case. Let us note that these two worst cases do not necessarily need to be the same cases (which is the reason that this property is called *weak* efficiency).

Strong Efficiency: *KDS* is strongly efficient if and only if the ratio of all processed events to processed external events in the worst case is small for any given motion.

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

Compactness: Kinetic data structures use a priority queue for event management. The length of the queue may be then expressed as a function of the size of the input data n. The data structure is compact if the maximum number of events ever present in the queue is small.

4.5 Combinatorial Analysis of Kinetic Data Structures

As described by [8], there is a significant difference between analyzing a discrete algorithm and a data structure behavior in the context of computational geometry. Consider for instance sorting algorithms as an example of discrete algorithms – it is easy to construct a sorting algorithm that runs in $O(n \log n)$ time over a set of general dataset of size n. There are sorting algorithms, that run in this amount of time in the worst case and furthermore, the performance of the algorithms is independent of the distribution of the sorted data. Using randomization, the average case can usually be transformed to the worst case [60]. It can be stated that the average case is completely subsumed by other measures – through randomization, any set of input data can be processed in $O(n \log n)$ time.

In computational geometry, the situation is different. For instance, in the case of convex hull construction over a set of n primitives in $d-$dimensional space, the computational complexity can be as high as $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ in the worst case [58, 76]. On the other hand, if the primitives are uniformly distributed inside a convex polytope, the expected asymptotic complexity of their convex hull construction is only $\Theta(n \log^{d-1} n)$ [12]. The difference between the average and worst case cannot be mitigated by changing the order in which the input data are processed or any other type of randomization; the complexity may differ significantly depending on the input data distribution.

The situation gets even more complicated when the context of kinetic data structures is considered, the main interest is to determine how many events (internal or external as defined earlier) will occur between time 0 and $+\infty$. Can there be infinitely many of them? In order to do that, one needs to start looking at the kinetic data structures from a different point of view. It can be observed that the problem of kinetic data structure analysis has some common features with one of the basic techniques in computational geometry – the *sweeping* algorithm.

4.5.1 Sweeping Algorithm

The very basic idea of the sweeping algorithm is to reduce a (static) $d-$dimensional problem to one less dimension by keeping track of $(d-1)-$arangements that change over time. An example of such approach would be the task of finding all intersections among a set of line segments S in a plane. In this case, a vertical sweep line is moving in the direction of axis x from left to right

over the line segments and one has to keep track of the $y-$coordinates of the segments that are currently intersecting the sweep line as shown in Fig. 4.1. Set S can be split into three different sets S_l, S_r, S_i for any value of x representing the current value of the sweep line. Sets S_l, S_r contain all the lines that are completely left or right from the sweep line respectively, S_i contains all the lines that are intersected by the sweep line.

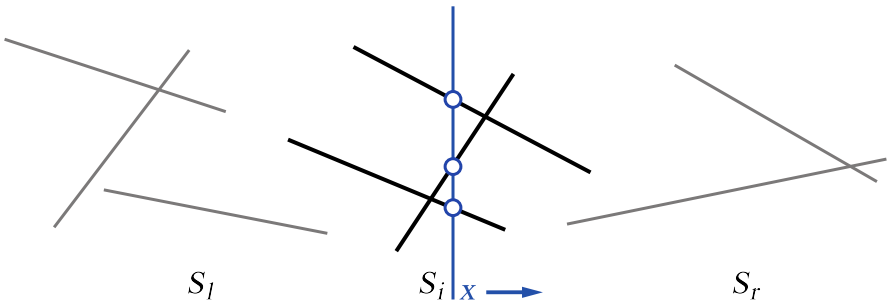


Figure 4.1: Sweeping sort algorithm principle.

Let x represent the current position of the sweep line, s_{min}, s_{max} are the minimum and maximum values of x of line segment $s \in S$, $s(x)$ is the y value of a line segment s at position x and up_s is the line segment immediately above s at the current value of the sweep line, we can formalize the algorithm in the following fashion [8]:

$$\begin{cases} \forall s \in S_l, & s_{max} < x \\ \forall s \in S_r, & s_{min} > x \\ \forall s \in S_i, & s_{min} < x \wedge s_{max} > x \\ \forall s \in S_i, & s(x) < up_s(x) \end{cases}$$

All of the comparisons stated above will become equal for some values of x. If the x coordinate of the sweep line is considered to be a time variable, these values of x will essentialy become *failure times* and the comparisons themselves will become certificate functions and the failure times will represent kinetic events.

This is very important observation because it allows a direct connection to be tied between the kinetic data structures and the space sweep paradigm. As mentioned in [8], the general nature of space sweep can be observed when one realizes that the convex hull diagram, closest pair, and Voronoi diagram problems can be solved in $O(n \log n)$ time using plane sweep methods [29].

4.5.2 Arangement of Curves

Since the connection between kinetic data structures and the plane sweep methods has been described, the fundamental question of this chapter can be addressed – how many events can be expected to happen in a kinetic data structure for any given interval of time. Let a lower envelope of functions and minimization diagram be defined as:

Definition 8 (Lower Envelope of Functions, [8])**.** *The lower envelope of a family $(f_i)_i$ of functions from $\mathbb{R}^d$ to $\mathbb{R}$ is the point-wise minimum:*

$$F(x) = \min_i f_i(x)$$

The *minimization diagram* is the combinatorial description of the lower envelope of such a family of functions, i.e., which function realizes the minimum at every point $x \in \mathbb{R}^d$. For univariate functions, this is simply a sequence of indices of the functions for individual subintervals of the axis x as shown in Fig. 4.2. For bivariate functions, this diagram is a planar map made of connected regions (faces, edges, and vertices) for which the minimum is realized by a fixed set of functions. Upper envelope and maximization diagram can be defined similarly.

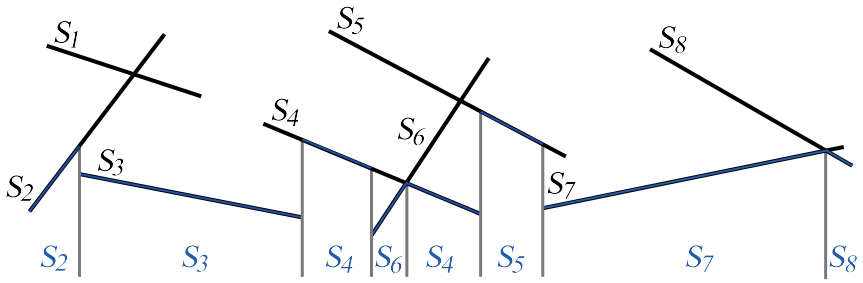


Figure 4.2: Minimization diagram.

Many problems in computational geometry can be converted into computing a lower or upper envelope using the principle of geometric duality which allows us to associate a point with a line. For example, the dual of convex hull diagram is a maximization diagram. Voronoi diagram of a set of points in d dimensions is a minimization diagram of a set of paraboloids in $d+1$ dimensions or a set of planes in $d+1$ dimensions.

It can be observed that the points where the function realizing the minimum in the minimization diagram changes, represent the kinetic events. The question itself can be formulated to estimate the worst-case complexity of constructing the minimization diagram and has been reduced into purely combinatorial task in the past. In order to solve this question, let us now define the Davenport-Schinzel Sequence:

Definition 9 (Davenport-Schinzel Sequence, [78])**.** *Let n,s be two positive integers. A sequence $\sigma = (\sigma_1, \ldots, \sigma_m)$ of integers is an (n,s)-Davenport-Schinzel sequence if it satisfies the following conditions:*

1. $\forall i \leq m : 1 \leq \sigma_i \leq n$
2. $\forall i < m : \sigma i \neq \sigma_{i+1}$
3. *there do not exist $s+2$ indices $1 \leq i_1 < i_2 < \ldots < i_{s+2} < m$ such that $\sigma_{i1} = \sigma_{i3} = \sigma_{i5} = \ldots = a$, $\sigma_{i2} = \sigma_{i4} = \sigma_{i6} = \ldots = b$, and $a \neq b$*

Where n is referred to as the number of symbols composing σ, s as the order of σ and $m = |\sigma|$ as the length of σ. The third condition basically states that long sequences of two alternating symbols are not allowed in Davenport-Schinzel sequences or in other words that the sequence is *non-repeating*.

Let $\lambda_s(n)$ be defined as the length of the longest (n,s)-Davenport-Schinzel sequence. To be able to compute the value of $\lambda_s(n)$, one has to define the Ackermann function first:

Definition 10 (Ackermann Function, [78])**.**

$$A(m,n) = \begin{cases} n+1 & if m = 0 \\ A(m-1,1) & if n = 0 \\ A(m-1,A(m,n-1)) & otherwise \end{cases}$$

Note that $A(n) = A(n,n)$ and let $\alpha(n)$ be the inverse function of $A(n)$. It has been shown in [78] that $\alpha(n) \leq 4$ for any practical values of n. For the purposes of this text, we will need the following values of $\lambda_s(n)$:

Theorem 1 (Values of $\lambda_s(n)$, [78])**.**

$$\begin{aligned} \lambda_1(n) &= n \\ \lambda_2(n) &= 2n-1 \\ \lambda_3(n) &= \Theta(n\alpha(n)) \\ \lambda_4(n) &= \Theta(n \cdot 2^{\alpha(n)}) \end{aligned}$$

Finally, if $(f_i)_{i=1}^n$ is a family of polynomials of degree up to s, the sequence of indices of the functions that realize its minimization diagram is a Davenport-Schinzel sequence as any two functions of f_i can intersect at most s times and therefore can alternate on the lower envelope at most $s+1$ times.

4.5.3 Kinetic Data Structure Events Analysis

As mentioned earlier in this text, the main interest in the analysis of kinetic data structures is to determine how many events can occur for a given data structure between time 0 and $+\infty$. The principle can be demonstrated on a very simple *KDS* – the closest pair. Consider a set of n points that move along linear trajectories in time with constant velocity. The task is to keep track of the single closest pair of points in this set. It can be easily seen that although the change of the closest pair is discrete, the distance between any two points changes continuously. If the distances of any two points in the set is plotted, $\binom{n}{2}$ curves will be obtained and the closest pair at any given time t is the pair that corresponds to the function at the lower envelope of these curves for t. Because the plotted curves describe the distance between pair of points, they can intersect at most twice (this fact can be seen easier if the quadratic distance of points is used instead of the normal distance). Changes in the closest pair then correspond with the vertices of the minimization diagram of this set of curves, the amount of which is $O(n^2)$ as per Theorem 1.

Many authors have examined this question in the past for different kinds of data structures, it has been shown in [6] that the closest pair changes $O(n\lambda_{2\delta}(n))$ times in a (δ,n)-scenario and there is a $(1,n)$-scenario in which the closest pair changes $\Omega(n^2)$ times. The term scenario in this context describes the basic properties of the moving points – in a (δ,n)-scenario, there is n points that move along polynomial trajectories of degree up to δ.

The same author has also shown that planar convex hull diagram changes $O(n\lambda_{2\delta}(n))$ times in a (δ,n)-scenario and there is a $(1,n)$-scenario in which the closest pair changes $\Omega(n^2)$ times.

Moving to some of the more complex data structures, it has been shown in [5] that the upper bound on the number of events for Delaunay triangulation (or Voronoi diagram, since the structures are mutually dual) is $O(n^d\lambda_s(n))$ using the same notation as in previous cases. The lower bound is the same as for convex hull, because convex hull is contained in the Delaunay triangulation as a subset. This topic will be discussed in more detail in Chapter 6.

4.5.4 Kinetic Data Structure Lifecycle

In order to explore the properties of external events in *KDS*, one needs to be able to describe the changes in the topology that are associated with them. To do that, let the explored data structures to be considered graphs $KDS(S) = \{S, E\}$ and extend the idea of graph *isomorphism* for the kinetic environment.

Note: Two graphs $G_1 = \{S, E_1\}, G_2 = \{S, E_2\}$ constructed over the same set S of points are isomorphic iff $\forall (u,v) \in E_1 \Leftrightarrow (u,v) \in E_2$.

Let $KDS(P,t)$ denote the state at time t of a kinetic data structure $KDS(P)$, constructed over a set of n time-dependent points $S = \{p_1(t), \ldots, p_n(t)\}$, where $\forall p_i \in S : p_i(t) = (x_i(t), y_i(t))$. Let $E = \{e_1, \ldots, e_m\}$ be a set of m edges in $KDS(S)$. Let $\forall e_i \in E : e_i = (p_k, p_l)$ if e_i is the edge defined by points $p_k, p_l \in S$. Finally, let $E(t)$ be the set of edges in a kinetic data structure at time t.

Definition 11 (Kinetic Equality)**.** *Given two* KDS *graphs* $G_1(S,t_1), G_2(S,t_2)$ *constructed over the same set of (time-dependent) points, the graphs are kinetically equal iff they are isomorphic.*

This relation will be denoted as $G_1(S,t_1) \cong G_2(S,t_2)$. In other words, two kinetic data structures constructed over the same set of points are kinetically equal if their topology is identical and the edges in those two data structures connect the same pairs of points. It is not necessary for the graphs to have equal values of time in order to be kinetically equal and therefore the (moving) points do not need to be at the same positions.

Let $\mathbb{E} = (t_1^{ex}, \ldots, t_n^{ex}), t_i^{ex} < t_{i+1}^{ex} \forall i < n$ be the set of all time values for which external events happen in $G(S,t)$. Formally, we can state that:

$$\forall t_i^{ex} \in \mathbb{E}\, \exists \varepsilon > 0 : G(S, t_i^{ex} - \varepsilon) \ncong G(S, t_i^{ex} + \varepsilon) \tag{4.3}$$

Eq. (4.3) describes the lifecycle of a *KDS* over a time interval $[t_1^{ex}; t_n^{ex}]$ and highlights the fact that external events change the topology of the graph as the set of edges changes with each external event. It is also worth mentioning that some of the states of $G(S,t)$ may be topologically equal to other states that existed in the graph for earlier values of t – this is the whole purpose of using the kinetic approach to handle data structures over sets of time dependent data. By computing the times of external topologic events, it is possible to determine every single different state of the data structure during its lifecycle without the risk of missing any of them. The internal events are more complicated to describe and shall be discussed later in this text with a concrete data structure to provide an example of their effect.

Note: The movement of points may affect multiple certificates at the same time. If such circumstances occur, it is vital to know that every single topologic change that is to happen in the data structure will be accompanied by a corresponding event and every single processed topologic event will result in one change in topology of the data structure. In the case of kinetic Delaunay triangulation, this means that no edge swap will cause the potential recursive swapping of nearby edges as in the case of some construction algorithms (e.g., the incremental insertion algorithm described in Chapter 3).

Chapter 5

Examples of Kinetic Data Structures Applications

The kinetic data structures (and kinetic Delaunay triangulations in particular) are a tool with a wide variety of possible uses, in this chapter we will describe some of their practical applications covering completely different areas and one general-use method. The kinetic Delaunay triangulation may be used as a tool for any object proximity based task, such as air traffic, simulation of animal groups of various kinds and might even be exploited for wide range of mathematical problems that may be described with systems of partial differential equations and solved by finite element method.

5.1 Collision Detection

Gavrilova proposed in [31] the concept of using the kinetic Delaunay triangulation as a means of collision detection between points in the generator set. The principle of this method lies in the fact that only such points may collide that share a common edge in the Delaunay triangulation. This is a consequence of the aforementioned duality of the Delaunay triangulation and the Voronoi diagram. Two colliding points are in fact a limit example of extremely close points. As shown in (2.1), the closer two points are, the more probable it is that their cells will neighbour in the Voronoi diagram. Together with the principle of duality, it may be stated that there will be a Delaunay edge connecting the two colliding points. Because all the points in the generator set may be moving, the edge between two colliding points will be present in the triangulation unspecifiably earlier than the collision occurs.

Erickson et al. showed in [26] that the kinetic approach may be used to detect collisions between convex polygons in $\mathbb{R}^2$. This work has later been extended by Guibas et al. in [40]. The later work then uses a kinetic regular triangulation, where the point weights are used to represent the radii of bounding spheres of the moving polyhedra.

Some practical applications of kinetic collision detection may include such applications as the one proposed by Goralski and Gold in [35] where the kinetic Voronoi diagrams are used to provide the spatial relations as an aid for human navigators of marine vessels – see Fig. 5.1. They may then use the provided information to improve their performance, as the main cause of accidents in this field is, according to the referred paper, the human error.

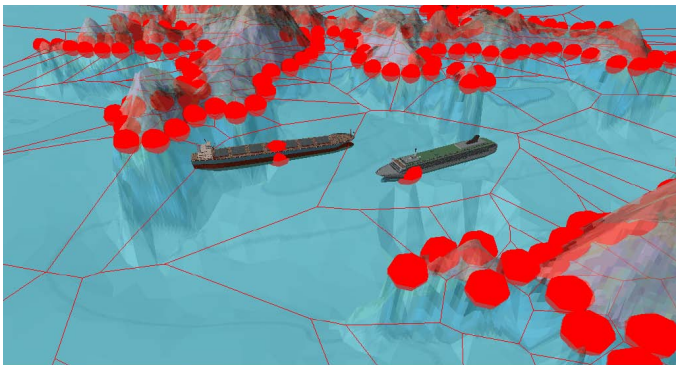


Figure 5.1: An example of the collision detection application of kinetic Voronoi diagram in the marine environment, [35].

Another practical use of the kinetic data structures (a kinetic regular triangulation in this specific case) is the work of Ferrez, see [27], where the kinetic regular triangulation is used to simulate movement of granular material (which incorporates the collision detection among the grains). Similarly as in the already mentioned work of Guibas et al. (see [35], the weights of the points in the simulation are used to represent the radii of the individual grains.

5.2 Simulation of Crowds

As presented by various research groups, the problem of crowd simulation (or other groups of autonomous agents such as different animal groups) may be handled in many different ways which are usually based on the adaptations of one of the basic approaches which include agent-based systems (examples may be found in [34, 47, 65, 84]) and potential based approach (such as the one presented in [43, 87]).

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

Goldenstein et al. shows in [34] that the local behaviour of the pedestrians may be handled by using a special kinetic data structure, which is in fact a special case of kinetic Delaunay triangulation. In this approach, each agent is assigned a "safe zone" and the kinetic data structure is used to keep track of the other agents that are nearby and may enter this safe zone. As we may see in Fig. 5.2, each agent (marked with the X in this figure) considers not only its nearest neighbours, but also some of their neighbours and so on if the concentration of agents is localy dense. In order to do that, it is necessary to add a new type of events to keep track of other agents entering or exiting the safety zone (agents inside the safety zone of X are shown as black circles).

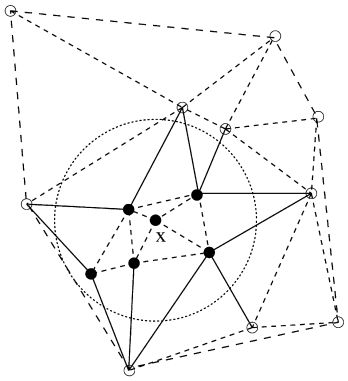


Figure 5.2: Crowd agent with a safe zone and connection to other nearby agents, [34].

5.3 Mathematical Simulations

The simulations of various different natural phenomena, such as fluid dynamics, requires to solve systems of partial differential equations. In practise, these equations are usually discretized and solved using finite difference method (*FDM*) or finite element method (*FEM*). Beni addresses the problem of solving partial differential equations in [11] in the field of Geographic Information System (GIS) simulations and suggests that the kinetic data structures (namely the kinetic Delaunay triangulation and kinetic Voronoi diagram) may be used for the purposes of fluid dynamic simulations in $\mathbb{R}^3$.

The Voronoi diagram is shown to be very well usable for the purposes of fluid representation as it is dynamic (incremental insertion algorithm is used for its construction) and thus allows mesh refinement in areas that need to be more detailed. Furthermore, the Voronoi cells are well-shaped and therefore provide good basis for the differential equations. To prove the usability of this approach, three different case studies are also presented in [11] – reflection of a compressible fluid inside a solid bounding cube where it is shown that the particles will compress upon reaching the walls of the bounding area, their velocity drops and their temperature, density and presure increases. The second case study shows one-directional expansion of gas in a tube; in this case the particles with higher density (which are in the middle of the tube) tend to move outwards and their volume increases. The last case study shows a fully three dimensional expansion of gas – the particles are initially pressurized in the centre of the simulation area and then begin to move outwards (see Fig. 5.3). Results from these case studies show that the kinetic Voronoi diagrams represent a valid choice for fluid dynamics simulations and are comparable with other commonly used approaches.

5.4 Motion Interpolation

The problem of motion interpolation is slightly different than the other presented applications of kinetic data structures as it uses the kinetic approach for handling purely dynamic problem (speaking in terms of triangulation updating techniques as shown in Chapter 4). An example would be a sampled process that is represented by different states of Delaunay triangulation for individual samples at $t = \{t_0, t_1, \ldots, t_n\}$ and it does not matter whether $t_{i+1} - t_i$ is constant or variable. In order to perform any sort of computation over the dataset at any given t_i, the

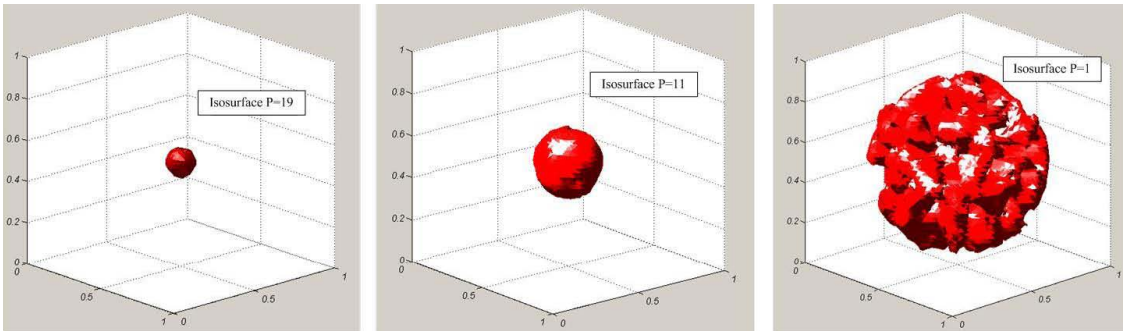


Figure 5.3: Simulation of gas expansion using kinetic Voronoi diagram, [11].

triangulation needs to be changed from the state at the time t_{i-1} to the state at $t = t_i$. As the sampling frequencies are usually quite high in practise, it may be expected that the states of the triangulation at t_i will be very similar to the state at t_{i+1} and thus it would be very convenient if this resemblance was exploited in any fashion.

Guibas and Russel showed in [36, 74] that this can be done by using the idea that only the states of the dataset at precisely given time instants are provided and nothing is known about the data between these times. The lack of information about the dataset inbetween the given times may even be exploited because the intermediate states will not be used for any kind of computation and may thus be manipulated in any viable fashion. The most straighforward method of transforming the dataset given at t_i to the dataset given at t_{i+1} would be to remove the points from the old triangulation and reinsert them back at the coordinates that correspond with the new state. To do so, one might either discard the whole old triangulation and recreate it from the scratch or remove and reinsert the points one-by-one. However, the referred works show that both of these ideas are slower than using the kinetic approach.

There are two different ways of moving the points from their old positions to the new ones with greater efficiency than rebuilding the whole triangulation. Either the trajectory travelled by each of the points or the polynomial degree in the certificate failure computation may be minimized. If the lengths of the point trajectories are to be minimized, they are moved along linear trajectories. If this approach is to be optimized a little further and it is possible to use the regular triangulation, the points might be moved along linear trajectories in the lifted space. The other approach, where the degrees of polynomials of the certificate functions are minimized, consists of moving the points along one axis at a time in the linear fashion.

Russel shows in his PhD. thesis [74] that the abovementioned methods themselves are not faster than the static updates due to the overhead of certificate failures computation, but they may be modified using a method called *filtering*. Filtering is based on the fact that a certain knowledge about the certificate functions might be used to determine whether any failures worth computing will occur or not; this is done by using methods similar to those mentioned in Chapter 4 such as the Descartes' rule of sign, Sturm sequences of polynomials and others. If the filtering is used, then the kinetic-based update techniques of the datasets outperform the static-based methods. A very similar filtering method was also presented in [104] with comparable results. The comparison of triangulation updates using the dynamic and kinetic approach not only shows the potential of the kinetic approach to be faster in the field of sets of moving data but also demonstrates its extraordinal versatility.

Part II

Theoretical Research

Chapter 6

Analysis of Selected Kinetic Data Structures

In this chapter, the kinetic Delaunay triangulation (*KDT*) and kinetic locally minimal triangulation (*KLMT*) are defined as kinetic data structures extension to their static counterparts. Their general properties are explored, as well as the existence of different types of kinetic events in both of these structures and the predicates and certificates associated with these events. The combinatorial properties of these data structures are also investigated, in order to establish the bounds on the number of events (both internal and external where they exist) which are processed during the lifecycle of these data structures. The results obtained in the analysis of both of these data structures are compared.

The introduction and analysis of *KLMT* represents a new research and has been published in [53, 54, 91].

6.1 General Arrangements

For the purpose of this chapter, kinetic data structures will be considered to be constructed over a set $S = \{p_1, p_2, \ldots, p_n\}$ of n time-dependent points so that $\forall p_q \in S:\ p_q(t) = (x_q(t), y_q(t))$ and the point movement will be limited so that $x_q(t), y_q(t)$ are polynomials. This restriction is quite common in kinetic data structure research as it provides a reasonably robust way of approximating more sophisticated trajectories and a good platform for further analysis and comparison with other kinetic data structures.

6.2 Kinetic Delaunay Triangulation

6.2.1 Events in Kinetic Delaunay Triangulation

If the points are allowed to move continuously by replacing their coordinates by functions of time, the certificates will become functions of time – *certificate functions*. In the case of the Delaunay triangulation, the only certificate validating the input data is represented by the incircle test as described in Chapter 2 – see Eq. (2.4). In this case, a determinant of a 4×4 matrix has to be computed in order to determine if a point lies inside, outside or on a circumcircle of any given

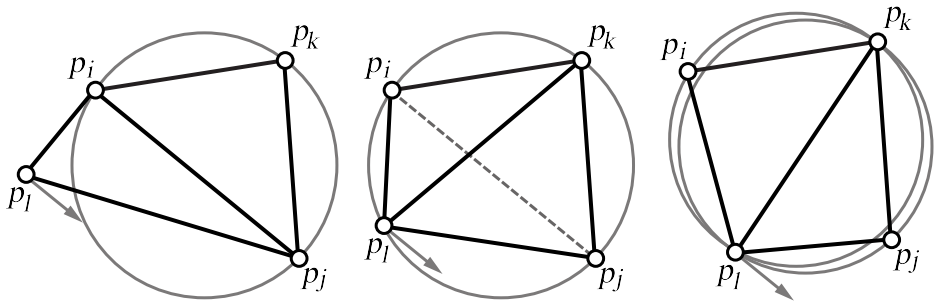
triangle (considering a planar triangulation). If the static points are replaced by moving points as defined in Eq. (4.1), a time dependent matrix incircle test is obtained in the following form:

$$incircle(p_i(t), p_j(t), p_k(t), p_l(t)) = \det \begin{bmatrix} x_i(t) & y_i(t) & x_i^2(t)+y_i^2(t) & 1 \\ x_j(t) & y_j(t) & x_j^2(t)+y_j^2(t) & 1 \\ x_k(t) & y_k(t) & x_k^2(t)+y_k^2(t) & 1 \\ x_l(t) & y_l(t) & x_l^2(t)+y_l^2(t) & 1 \end{bmatrix} \tag{6.1}$$

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

Note: If polynomial functions are used as the coordinate functions, the certificate function becomes a polynomial function itself. If only linear functions are used for the coordinates, the certificate function becomes a polynomial of the fourth or lower degree (in this case, the points move along linear trajectories without any acceleration).

An example of topologic event in *KDT* is given in Fig. 6.1. In Fig. 6.1(a), a point marked as p_l moves inside the circumcircle of a static triangle $p_ip_jp_k$ (none of its points is moving). As p_l reaches the circumcircle of $p_ip_jp_k$ (shown in Fig. 6.1(b)), the assigned certificate function reaches a zero value which causes the certificate to fail. In order to keep the Delaunay property, an edge swap has to be performed, which is done by replacing the edge p_ip_j by the edge p_kp_l, as depicted in Fig. 6.1(c).



(a) Initial situation. (b) Topologic event. (c) Edge swap is performed.

Figure 6.1: Topologic event in the Delaunay triangulation.

There are only topologic events of this type in *KDT* since it is validated simply by checking if every point lies outside the circumcircles of all the triangles in the triangulation, no other condition needs to be checked and therefore, no other certificate functions exist for this particular kinetic data structure.

Since the topologic event in the Delaunay triangulation is an external event as defined earlier in Chapter 4, its direct result is a change in the topology of *KDS* (the aforementioned edge swap). An important consequence of this change in topology is that two triangles are destroyed that have existed in the triangulation until the time of the event and two other triangles are created that did not previously exist. It is important to compute if any topologic events should be scheduled for the new triangles and deschedule any events that might still be scheduled for the triangles being destroyed as outlined in Alg. 3.

6.2.2 General Properties of Kinetic Delaunay Triangulation

The kinetic data structures properties may be used to evaluate the *KDT* (similar evaluations may be found in [10,37], where the properties of *KDT* and other kinetic data structures are explored). Using the knowledge from the previous text, it may be stated that *KDT* has the following properties:

***KDT* is responsive:** Each event in *KDT* will create two new triangles, which means that up to five new events need to be scheduled – one for the edge between the new triangles and then one for each of the triangles and its two other neighbors. Therefore, managing an event takes almost constant time for any event.

***KDT* is strongly efficient:** Only one type of events exist in *KDT*, and thus all the events are external. The ratio of all processed events to processed external events is $O(1)$.

***KDT* is not local:** Each moving point may be connected to a relatively large number of neighbours and thus participate in a relatively large number of events.

***KDT* is compact:** Although the precise relations between the size of the input set, the allowed type of movement and the size of the event queue are not known, the compactness of *KDT* has been shown – see [5,37].

6.2.3 Bounds on the Number of Events in Kinetic Delaunay Triangulation

As discussed in [5,39,73], there are certain properties of the topologic events in *KDT*, that can be used to estimate the upper bound on the number of events that will occur in its lifecycle. First of all, every topologic event is connected with a subset $S' \subset S$ of size $d+2$, where d is the dimension of the data structure. In the case of planar *KDT*, the events are tied to a pair of adjacent triangles. For the computation of the bounds, only certificate functions with finite number of roots are allowed (such as polynomials of bounded degree up to r). This assumption implies that each quadrilateral formed by two adjacent triangles generates at most a constant number of events and provides an overall upper bound of at most$r\binom{n+1}{d+2} \subset O(n^{d+2})$ on the total number of events.

Using the Davenport-Schinzel argument, this upper bound can be improved by approximately a linear factor to $O(n^2\lambda_s(n))$. For points moving along linear trajectories, the maximum number of roots of each certificate function is 4. According to Theorem 1, the value of $\lambda_4(n) = \Theta(n \cdot 2^{\alpha(n)})$ this provides an upper bound of $O(n^3 \cdot 2^{\alpha(n)})$.

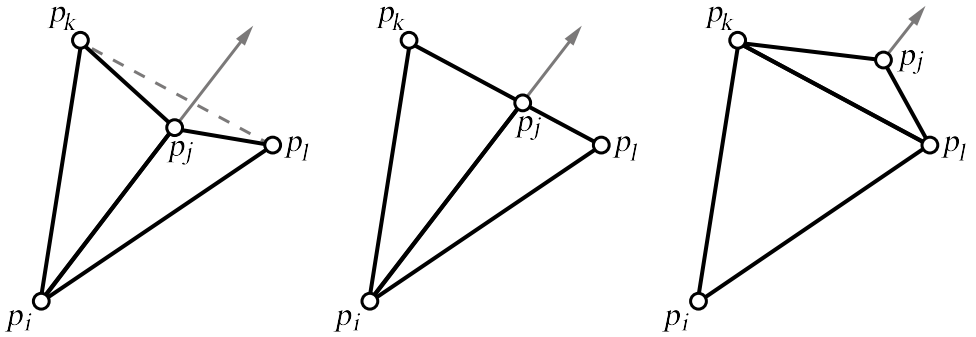
It was also shown in [3] that the convex hull of a moving point set may change $\Theta(n^2)$ times, which implies a lower bound of $\Omega(n^2)$ topologic events in any triangulation constructed over a set of moving points.

6.3 Kinetic Locally Minimal Triangulation

6.3.1 Events in Kinetic Locally Minimal Triangulation

As described in the previous text, there are two predicates that need to be validated for *KLMT* – the local minimum as defined in Eq. (2.7) and the local convexity as defined in Eq. (6.2). Each

of those certificates will provide one predicate and by extension one type of event which are illustrated in Fig. 6.2.

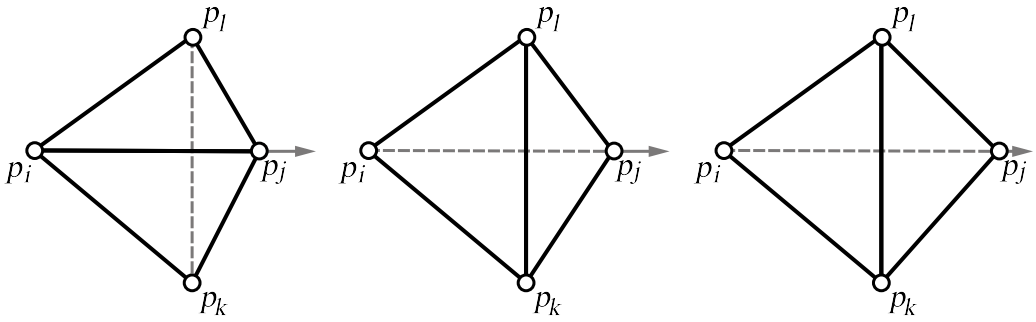


(a) Initial configuration. (b) Colinear event. (c) Local minimum event.

Figure 6.2: Events in *KLMT*.

In this example, two triangles forming $p_i p_j p_k$ and $p_i p_l p_j$ are shown that share a common edge $p_i p_j$ and form a non-convex quadrilateral. Point p_j moves along a linear trajectory as shown by the gray arrow. The edge $p_k p_l$ is clearly shorter than the edge $p_i p_j$, but it has been described in the previous text that the common edge shared by the two triangles cannot be swapped unless the quadrilateral is convex.

As soon as p_j reaches the position shown in Fig. 6.2(b), points p_j, p_k, p_l become colinear and the quadrilateral ceases to be non-convex. This denotes a so called a *colinear event*. Note that the edge still cannot be swapped because the quadrilateral is now in a singular state and the other possible edge $p_k p_j$ should formally replace $p_i p_j$ immediately after p_j moves away from the line $p_k p_l$. The result of this edge swap is shown in Fig. 6.2(c). The swap itself is a result of a *topologic event*.



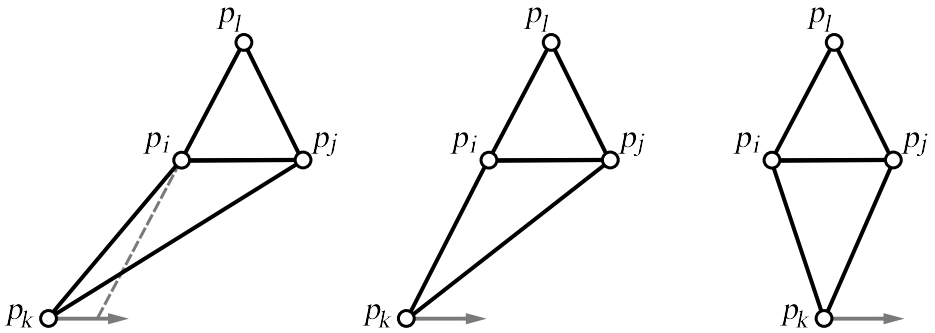
(a) Initial configuration. (b) Topologic event. (c) Final configuration.

Figure 6.3: *KLMT* with topologic events only.

The colinear event is an internal event because it does not change the topology of *KLMT* but is necessary to be processed in order to schedule and de-schedule the topologic events, which on the other hand are *external events* because they result in topology changes represented by the edge swaps as described above. Generally, the two types of events are not directly tied together and it is not necessary for the edge swap to follow immediately after the colinear event and in some configurations, it will not happen at all. The events may occur in the triangulation totally independent of one another. Perhaps the most basic example of this fact is shown in Fig. 6.3,

where such a *KLMT* is presented, in which only topologic events occur. In this example, the initial triangulation shown in Fig. 6.3(a) reaches a singular state shown in Fig. 6.3(b) because of the movement of p_l along the linear trajectory marked by the grey line and an edge swap is performed in order to maintain the local minimality condition, resulting in the configuration shown in Fig. 6.3(c). It is obvious that no more events of any kind will be scheduled for the future movement of p_l.

It is also possible to construct such cases of *KLMT* that will schedule and execute only internal events. Example of such a triangulation can be seen in Fig. 6.4. In this particular example, the length of $p_i p_j$ is shorter than the length of $p_k p_l$ for any value of time. A colinear event will be scheduled but no topologic change will ever occur in the triangulation.



(a) Initial configuration. (b) Colinear event. (c) Convex quadrilateral.

Figure 6.4: Non-convex configuration without external events.

Fig. 6.5 shows another special case of possible event configuration in *KLMT*. In the initial state, the length of edge $p_i p_j$ is shorter than $p_k p_l$ and the quadrilateral is non-convex. Due to the movement of p_j, the two edges become of equal length when p_j reaches the position of p'_j and finally $p_i p_j$ becomes longer than $p_k p_l$, represented here by the position of p''_j. During the whole lifecycle of this triangulation example, the quadrilateral remains non-convex and therefore, no topology change can be performed on the triangulation even though a topologic event would be scheduled at the time when $p_j = p'_j$.

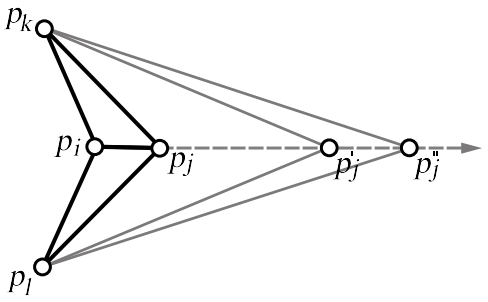
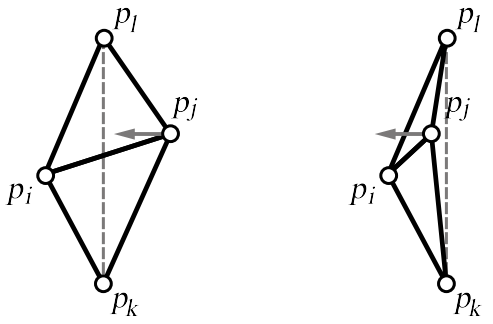


Figure 6.5: Configuration of *KLMT* without colinear events.

Colinear Events Computation

It is important to note that the convexity certificate as described in Eq. (2.8) is simplified to provide the result using the least possible number of mathematical operations. This makes it

optimal for the use in static *LMT* construction, however, it also makes it insufficient for the use in *KLMT*. Consider the triangulation in Fig. 6.6 – in this example, there are two adjacent triangles $p_i p_j p_k$ and $p_i p_j p_l$ forming a convex quadrilateral with the common edge (the shorter diagonal) being $p_i p_j$ and point p_j moves towards point p_i.



(a) Initial configuration. (b) Non-convex state.

Figure 6.6: Special case of the convexity certificate computation.

It can be observed that the value of Eq. (2.8) will not change its sign after the quadrilateral becomes non-convex, because the orientation of both of the tested triangles $p_k p_l p_i$ and $p_k p_l p_j$ does not change as a result of the movement. Because of this fact, a different convexity certificate needs to be introduced for *KLMT* as shown in Eq. (6.2):

$$CH'(S_q) = orient(p_i, p_j, p_k) \cdot orient(p_l, p_k, p_i) \cdot orient(p_j, p_i, p_l) \cdot orient(p_k, p_l, p_j) \tag{6.2}$$

6.3.2 General Properties of Kinetic Locally Minimal Triangulation

The general kinetic data structure properties of *KLMT* will be explored as defined in Chapter 4. Let the *responsiveness* and *compactness* be considered first.

Lemma 1. KLMT *is responsive.*

Proof. There are two different types of events in *KLMT*: *colinear* and *edge swap.* If *KLMT* is responsive, handling them will take only a small amount of time:

Colinear: Since this event is internal, it will not directly affect the data structure in any way. However, if the affected quadrilateral was non-convex prior to this event, it becomes convex and therefore, valid for topologic events. Because of that, up to one topologic event needs to be scheduled for the newly convex quadrilateral. And vice versa if the quadrilateral was convex prior to the colinear event, up to one topologic event needs to be descheduled, that might have been scheduled for the quadrilateral.

Edge Swap: This is an external event because swapping the inner edge in a convex quadrilateral will change the topology of *KLMT*. The result of the edge swap is up to 5 new quadrilaterals being formed in the triangulation (one is created by the edge swap and the other four may be created from the newly created triangles and the triangles directly adjacent to the quadrilateral). For each of those quadrilaterals up to one new event needs to be scheduled

(either colinear or topologic, depending on the actual current state of the quadrilateral). Also, up to four events may be descheduled, because the quadrilaterals that were used to compute them do not exist any more as a result of the edge swap.

Each of the aforementioned operations is independent of the total number of points in the triangulation, therefore, the amount of time required to handle the events in *KLMT* is $O(1)$. Also, the topologic changes performed as a result of handling an external event are local and will only affect the related quadrilateral. □

Lemma 2. KLMT *is compact.*

Proof. To prove that *KLMT* is compact, it must be shown that the number of events ever present in the queue is small. There is only a constant number of events that can be scheduled for any given edge present in the triangulation represented by the roots of the certificate function for the given quadrilateral. When an edge is removed from the triangulation as a result of handling a topologic event, the connected events are also removed from the event priority queue. And since the number of edges in $KLMT(P)$ is $O(n)$ at any given time, *KLMT* is compact. □

Regarding the efficiency of *KLMT*, it is quite simple to show that *KLMT* is not *strongly* efficient even for polynomial trajectories of the points. The strong efficiency condition requires the aforementioned ratio to be small for *any movement* – consider the example shown in Fig. 6.7. Point $p_l = (a \cdot \sin(b \cdot t + c), d)$, where $a, b, c, d = const.$, moves in a similar way as shown in Fig. 6.4 with the exception that once it reaches position p'_l, its direction of movement is reversed and it starts moving back to its original position along the same trajectory, with the same velocity. Therefore, generating four colinear events per each completion of its trajectory cycle and no topologic events are ever scheduled during the triangulation lifecycle.

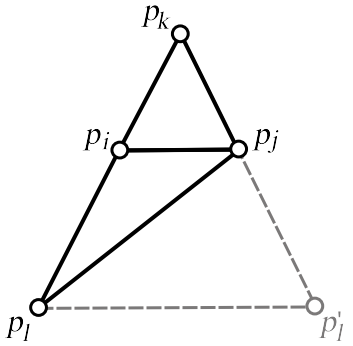


Figure 6.7: Periodic type of movement in *KLMT*.

Lemma 3. KLMT *is not strongly efficient.*

Proof. As shown in Fig. 6.7, it is possible to construct such a *KLMT*, where colinear events occur, but no topologic event is ever scheduled. Therefore, the ratio of all processed events to processed external events cannot be small, since there are no processed external events. □

Our previous work in [91] stated that *KLMT* is strongly efficient, however in the subsequent research, Martin Maňák suggested that the opposite is in fact true and this data structure is weakly efficient, which was proved, as stated in this text. Nevertheless, if the point movement

is strictly limited to polynomial trajectories, the resulting *KLMT* is indeed strongly efficient as shown in [91], the proof of strong efficiency in this limited case is identical to the proof of *KLMT* weak efficiency in this thesis, see further.

Before the weak efficiency of *KLMT* can be evaluated, the worst case number of external events this data structure can process during its lifetime has to be determined. Note that $\lambda_s(n)$ is the maximum length of (n,s) Davenport-Schinzel sequence (see [5]) as defined in the previous text.

As mentioned in the previous text, [5] shows that number of topologic events in a kinetic Delaunay triangulation in *2*-dimensional Euclidean space is bounded by $O(n^2\lambda_s(n))$, if each subset of P of size 4 generates at most a constant number s of external events. The same proof is also valid for *KLMT*: since the movement of the points in P is limited to polynomial trajectories, Eq. (2.7) will be a polynomial and therefore, provide a constant number of roots (which correspond with the topologic events). Similarly to *KDT*, the topologic events are also bound to a single quadrilateral and leave its bounding edges unchanged.

Corollary 1. *The upper bound on the total number of external (edge swap) events in KLMT*(P) *is* $O(n^2\lambda_s(n))$.

Theorem 2. *The number of internal (colinear) events processed by KLMT*(P) *is bounded by* $O(n^2\lambda_s(n))$.

Proof. As stated in the previous text, internal events do not change the topology of the kinetic data structure by their definition. Therefore, if the number of possible internal events on each quadrilateral is at most a constant number, the total number of internal events is bounded by the number of external events in the data structure because new quadrilaterals can only be formed as a result of handling external (edge swap) events. Eq. (6.2) shows that the number of internal events for any given quadrilateral is equal to the number of roots of a polynomial equation, therefore, independent of the size of P and thus constant. □

Lemma 4. KLMT *is weakly efficient.*

Proof. Theorem 2 shows that the upper bounds on internal (colinear) and external (topologic) events is $O(n^2\lambda_s(n))$. Corollary 1 shows that the number of external events in *KLMT(P)* is bounded by $O(n^2\lambda_s(n))$. The ratio between the number of all processed events to the number of processed external events can be written as follows:

$$\frac{O(n^2\lambda_s(n)) + O(n^2\lambda_s(n))}{O(n^2\lambda_s(n))} = \frac{O(n^2\lambda_s(n))}{O(n^2\lambda_s(n))}$$
$$\frac{O(n^2\lambda_s(n))}{O(n^2\lambda_s(n))} \subset O(\log^{\varepsilon} n)$$

The ratio is *small* as defined in the previous text and therefore, *KLMT(P)* is weakly efficient. □

Lemma 5. KLMT *is not local.*

Proof. Consider the triangulation in Fig. 6.8 - all the points in *KLMT* but q are placed on a circular arc. The triangulation is locally minimal and no edge swaps can be performed because the quadrilaterals containing q are non-convex, yet q is connected to all other points in P, the number of which is bounded by $\Theta(n)$. Therefore, q is included in every single certificate that will occur in $KLMT(P)$ and any change of q will affect all the enqueued certificates.

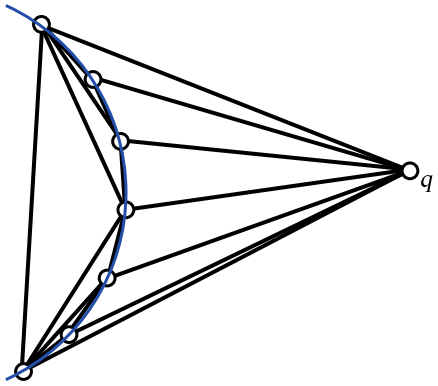


Figure 6.8: Non-local case of *KLMT*.

□

6.4 Comparison of Kinetic Delaunay Triangulation and Locally Minimal Triangulation

The comparison of the basic *KDS* properties is shown in Tab. 6.1. Both of these structures are responsive and compact, neither is local and while *KDT* is strongly efficient, *KLMT* is only weakly efficient.

Property	***KDT***	***KLMT***
Responsiveness	yes	yes
Efficiency	strong	weak
Locality	no	no
Compactness	yes	yes

Table 6.1: Comparison of *KDT* and *KLMT* basic properties.

Using the apparatus published in [78], it can be shown that the bounds on the number of events processed during the lifecycle of *KLMT* are $O(n^d \lambda_s(n))$, similarly to the case of *KDT*. The certificates used to compute the events are generally simpler in the case of *KLMT* which results in tighter bounds on the number of events. For linear trajectories in $\mathbb{R}^2$, the total number of events in *KLMT* is bounded by $O(n^3)$ which is smaller for any values of n than $O(n^3 \cdot 2^{\alpha(n)})$ which is the respective bound for the number of events in *KDT*. All the discussed events can be handled in $O(1)$ and therefore, the management of *KLMT* should be computationally simpler than the management of *KDT*. However, the management of *KDT* only requires handling of one type of external events, while the management of *KLMT* requires us to handle internal events as well and this may lead to increased numerical instability.

Chapter 7

Event Time Computation

According to our research [93], the event computation in kinetic data structures can represent the most time consuming part of the whole *KDS* management process by a large percentage. Depending on the ratio of moving points to static points in the generating point set, the task of event time computation can consume as much as roughly 95 % of the time. This chapter will describe some of the methods used for the computation of events and focus on their properties regarding the event classification. Later, a method will be shown, how the potential events can be classified ahead of time in order to avoid the computation of events that will never be executed. Without any loss of generality, the aforementioned topics will be described on the example of kinetic Delaunay triangulation. The research presented in this chapter has been published in [92, 94, 95].

7.1 Introduction

The kinetic data structures and especially the kinetic Delaunay triangulation have been undergoing an extensive research in the past years. Result of this is that their behavior is relatively very well understood, especially in their planar variants (there is an ongoing research in higher dimensions [11, 56, 74]. One of the main issues in this field (and perhaps the most important one) is the way of certificate failures (i.e., events) computation. Various methods are presented with the modern trend being the use of interval approach [74] or a speed-optimized numerical solvers [11]. Each of these methods is usable in different conditions – the numerical solvers are usually quicker for polynomials of low degree, whereas the polynomials of degree greater than six are better solved by the interval-based methods. This feature also partially determines the field of use of these methods. The numerical solvers are more likely to be found in the applications where the continuity of the movement does not play a key role and it is possible to simulate curved trajectories by piecewise linear curves (points moving along linear trajectories generate low order polynomials as certificate functions). On the other hand, if the movement is to be smooth, a more sophisticated methods are likely to be used which will ensure better runtime consumption for higher order polynomials.

7.2 Event Computation Equations

As stated before, for several reasons it is often convenient to restrict the allowed point movement to polynomial functions of time, let the moving points as shown in Eq. (4.1) be defined as such – i.e., with the point coordinates $x_i(t), y_i(t)$ being polynomial functions of variable $t \in \mathbb{R}$. The certificate functions then becomes a polynomial:

$$c(t) = incircle(p_i(t), p_j(t), p_k(t), p_l(t)) = \sum_{i=0}^{n} a_i \cdot t^i \tag{7.1}$$

where $incircle(p_i(t), p_j(t), p_k(t), p_l(t))$ is the certificate function for *KDT* as shown in Eq. (6.1), n is the degree of the resulting polynomial, $t \in \mathbb{R}$ is the time variable and $a_0, \ldots, a_n$ are the coefficients of the polynomial. It can be seen that n satisfies the following condition:

$$n \leq 4 \cdot \max_{q=i,j,k,l} \{\deg p_q(t)\}$$

Specifically, for linear point trajectories, $n \leq 4$ as the certificate function becomes a polynomial of degree no greater than four. In order to obtain the times of the events associated with p_i, p_j, p_k, p_l, the roots of $c(t)$ have to be found. Some of the methods for solving these polynomials will be discussed in this chapter.

7.2.1 Event Computation Methods Overview

This section provides an overview of the most commonly used methods for polynomial root location in the context of the kinetic data structures. According to [38], these methods may be divided into three basic categories:

Naive approach: the most straighforward method consists of computing all the real roots of the given polynomial, discarding the roots that are not usable (e.g., all the negative roots) and enqueuing new events that occur at the times determined by the remaining roots (if there are any left). The methods used for computing these roots may include such approaches as *Laguerre method*, *eigenvalue-based solvers*, *Newton method*, *Bairstow method* and other numerical methods or their combination. Details on variety of these methods may be found in literature – see [25, 67, 71] and others.

Interval based methods: first proposed in [38], these methods represent a different approach. Instead of computing the polynomial roots, intervals are used that contain only one root each. The intervals may be of virtually any size when they are computed and they are reduced when the need to do so emerges – for instance when they are stored in the priority queue or it needs to be determined which root certificate should be handled first. The main advantage of this approach is the fact that not each root becomes computed exactly and runtime can be saved by not computing the roots that are not really needed (the events may become de-scheduled as explained later in Section 7.5). Details on this approach may be found in [38, 74].

Hybrid methods: using algebraic preliminaries such as the Descartes' rule of sign and the Sturm sequences of polynomials – see Section 7.4, these methods are able to divide the real axis into intervals containing the individual roots of a given polynomial. Using some

iterative approach (such as the Newton method or bisection), it is possible to determine the location of the roots. In order to be able to employ these iterative methods efficiently, the polynomials may be replaced by other polynomials with exactly the same roots but with multiplicities equal to one (see [38]). Details on these methods may be found in [38,72,74].

Guibas and Karavelas also provide a comparison of some representants of the aforementioned methods in [38], where they show that both the hybrid and the interval based methods are better than the numerical solvers (in the terms of runtime consumption) for polynomial trajectories, where the certificate function result in a polynomial of degree greater than six. For the lower degree polynomials, the numerical methods are quicker.

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

7.2.2 Polynomial Solving

In the previous text, the term polynomial has been used informally, in order to explore the methods for polynomial root location, it is necessary to lay down some basic definitions:

Definition 12 (Polynomial)**.** *Polynomial represents a special case of nonlinear equation:*

$$f(z) \equiv a_n z^n + a_{n-1} z^{n-1} + ... + a_1 z + a_0 = 0 \tag{7.2}$$

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, let $z \in \mathbb{R}$, thus making the polynomial in Eq. (7.2) a real polynomial of n-th degree.

Definition 13 (Monic Polynomial)**.** *A polynomial of degree n is monic iff* $a_n = 1$.

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

Definition 14 (Polynomial root, root 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.

Polynomial Root Count

As a result of the fundamental theorem of algebra [100] applied to polynomials with real coefficients, it can be stated that the number of the complex roots of such a polynomial will be either zero or even.

Root Position Bounds

Given a polynomial equation as shown in Eq. (7.1), according to [44] the value of each of its real roots z_i may be restricted as follows:

$$|z_i| = \max\left\{1, \frac{1}{a_n}\sum_{i=0}^{n}|a_i|\right\} \tag{7.3}$$

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

Descartes' Rule of Sign

This simple rule (discussed for instance in [72]) can be used to simply obtain an upper bound on the number of the real roots of a polynomial on any given interval:

Theorem 3 (Descartes' Rule of Sign)**.** *Let $c(t)$ be a polynomial as in Eq.* (7.1)*. Let $V(c)$ be the number of sign variations in the sequence $(a_0, \ldots, a_n)$, discarding all zero values, and $\|c_+\|$ the number of all the positive roots of $c(t)$ with multiplicities, then $\|c_+\| \leq V(c)$ and $V(c) - \|c_+\|$ is even.*

Using the transformations of paramater t as shown in [72], the number of real roots can be determined for any given interval.

7.3 Analytical Methods for Solving Polynomials

7.3.1 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.

7.3.2 Analytical Formulas

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

$$a_1 z + a_0 = 0$$

may be found using the following formula:

$$z = -\frac{a_0}{a_1} \tag{7.4}$$

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. The quadratic equation can be defined 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:

$$x_1, x_2 = -b \pm \frac{\sqrt{D}}{2 \cdot a_2} \tag{7.5}$$

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):

$$z^3 + a_2 z^2 + a_1 z + a_0 = 0$$

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 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 [99]. 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: To solve the quartic equation, defined as

$$z^4 + a_3 z^3 + a_2 z^2 + a_1 z + a_0 = 0$$

the Vietta's formula can be used, described for instance in [101], but the same restrictions as in the previous case apply. The subresults need to be stored as complex numbers even if the roots themselves are real, which may theoretically decrease the precision of the final results.

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

7.4 Numerical Methods for Solving Polynomials

7.4.1 General Methods for Solving Nonlinear Equations

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 $[x_1, x_2]$ and a function $f = f(x)$ such that f is continuous at $[x_1, x_2]$ and

$$f(x_1) \cdot f(x_2) < 0 \tag{7.6}$$

then, due to the law of the mean, at least one root of f lies in $[x_1, x_2]$.

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 Eq. (7.6).

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 [71]).

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 $[x_1, x_2]$. The progress of Regula falsi is illustrated in Figure 7.1.

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 [71]).

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 7.2.

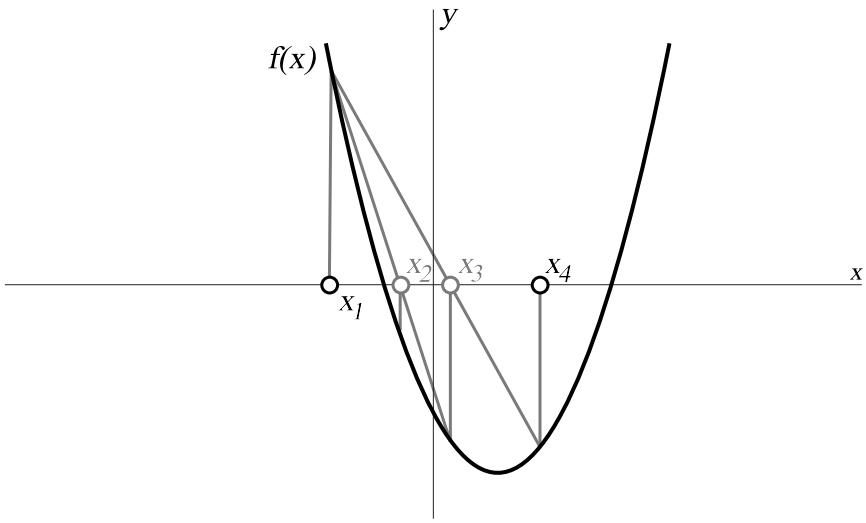


Figure 7.1: The first few iterations of Regula falsi method.

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:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{7.7}$$

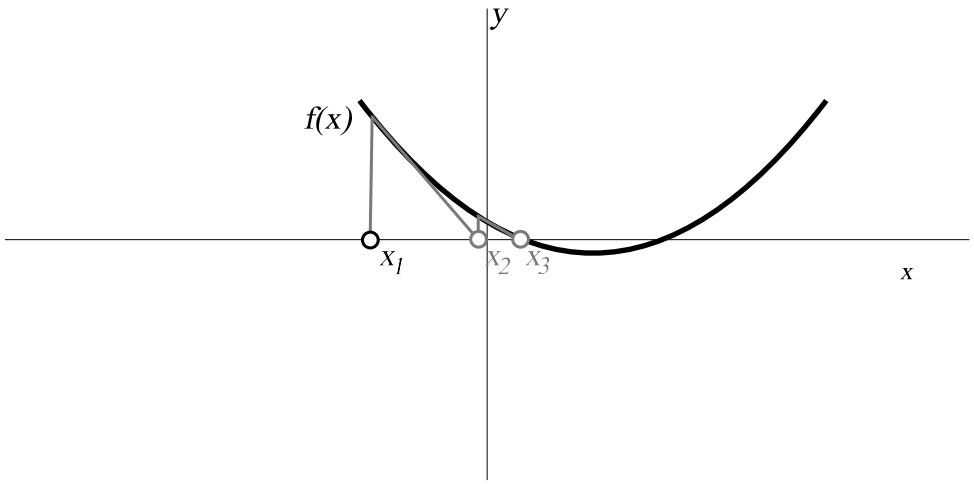


Figure 7.2: The first few iterations of Newton-Rhapson method.

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.

7.4.2 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 [66, 71].

7.4.3 Sturm Sequences of Polynomials

Definition 15 (Sturm Sequences, [71])**.** *The sequence of polynomials*

$$f_1(x), f_2(x), \ldots, f_m(x)$$

will be Sturm sequence at interval $[a,b]$ *(a and b may be infinite), if:*

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

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

The key feature of the Sturm sequences of polynomials is that they may be used to compute the total number of polynomial roots on any given interval including their multiplicities. Consequence of this fact is that Sturm sequences may be quite easily used for polynomial root computation. Sturm sequence of a polynomial $f(x)$ may be constructed [71]:

$$\begin{aligned} f_1(x) &= f(x) \\ f_2(x) &= f'(x) \\ f_{j-1}(x) &= q_{j-1}(x) f_j(x) - f_{j+1}(x), j = 2,...,m-1 \\ f_{m-1}(x) &= q_{m-1}(x) f_m(x) \end{aligned} \tag{7.8}$$

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 $f_j(x)$. Therefore, $\{f_i(x)\}$ is a sequence of polynomials of a decreasing degree. The first term of the sequence is the input polynomial, the second term 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 if the third equation from Eq. (7.8) is rewritten to the following form:

$$\frac{f_{j-1}(x)}{f_j(x)} = q_{j-1}(x) + (-1) \cdot \frac{f_{j+1}(x)}{f_j(x)} \tag{7.9}$$

A division of two polynomials can be seen in Eq. (7.9), with $f_{j-1}(x)$ being the numerator and $f_j(x)$ being the denominator of the division. $q_{j-1}(x)$ then denotes the quotient (which is not used for the practical computation of the Sturm sequences) and $f_{j+1}(x)$ is the negation of the remainder of the division (the multiplication of $f_{j+1}(x)$ by the constant -1 is necessary to make the relation correct.

Important Features of Sturm Sequence of Polynomials

Counting the Roots: Let $V(x)$ be the number of the sign changes in the Sturm sequence Eq. (7.8) (ignoring all zeros). This function may then be used to count the number of distinct real roots of $f(x)$ on any interval $[a,b]$:

$$r_{a,b} = V(a) - V(b) \tag{7.10}$$

where $a,b \in \mathbb{R}$ or either of a,b may be infinite. As proved in [71], Eq. (7.10) remains valid even if a or b are the roots of $f(x)$.

Root Multiplicity: The last term of the Sturm sequence Eq. (7.8) may be used to distinguish and compute the values of the multiple roots of $f(x)$. As proved in [71], all the multiple roots of $f(x)$ with multiplicities decreased by one are the roots of $f_m(x)$, which does not have any other roots. Together with the fundamental theorem of algebra, this statement may be extended to various useful conclusions. For instance if $f_m(x)$ is of an odd degree, then $f(x)$ has at least one multiple root, etc.

Generalized Sturm Sequence

If the initial polynomial has some multiple roots, the created sequence is no further a Sturm sequence as defined in Def. 15, 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 this text, a generalized Sturm sequence is meant.

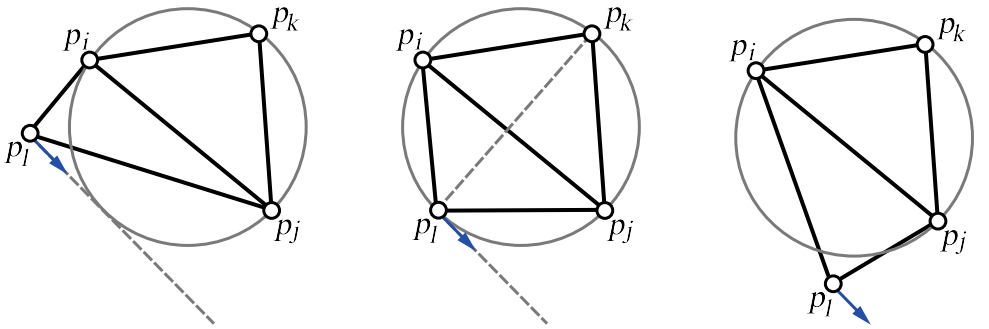
7.5 Event Classification and Redundancy

Generally speaking, there are two reasons why the root of a certificate function would not describe a to-be-executed event in the kinetic data structure. In some cases, the root might point to a specific topologic situation that does not require an event to be executed, while in other cases, the execution of the event might become unnecessary because it is scheduled too far in the future and the kinetic data structure will enter such a state that prevents the event from execution due to some other events executed earlier and the event becomes obsolete.

7.5.1 Insignificant Polynomial Roots

Consider the situation depicted in Fig. 7.3. In this figure, the point p_l moves tangentially to the circumcircle of the triangle $p_i p_j p_k$.

If the points p_i, p_j, p_k are static there is only one certificate failure which is displayed as a singular state in Fig. 7.3(b). If an event is scheduled for this certificate failure, the triangulation will cease to be Delaunay and eventually it would even cease to be a triangulation as defined earlier because the edges would start to overlap. It is thus necessary to distinguish this type of events.



(a) Initial situation.

(b) Singular state.

(c) Point p_l moves forward without any changes in the topology.

Figure 7.3: Tangential movement of a point against the circumcircle of a triangle.

Note: In accordance with Eq. (4.3), it can be seen that during the whole depicted interval of p_l movement, the graph does not change its topology and therefore no external event could have occured.

The situation shown in Fig. 7.3 is the simplest possible case of the occurence of this type of event and it may distinguished by obtaining a double root of the certificate function. The following lemma shows how the similar situations can be detected for more complicated point configurations as long as the movement of the points remain polynomial.

Lemma 6. *The roots of a polynomial certificate function $c(t)$ may be divided into two groups as follows:*

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

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

$$c(t) = (t - t_0)^r \cdot q(t) \tag{7.11}$$

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

If t_0 is of even multiplicity, then $r = 2k$ and thus:

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

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

If t_0 is of odd multiplicity, r can be expressed as $r = 2k+1$ and c_0 can be expressed as:

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

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

As a result of the presented knowledge, it can be stated that any method suitable for an easy determination ot the polynomial roots multiplicity is highly valuable. Also, the hybrid method presented by Guibas and Karavelas in [38] is not entirely correct. Prior to the actual root computation, their method replaces each certificate function $c(t)$ with other polynomial function $d(t)$ which has the same roots but all with multiplicities equal to one. As shown in the example in Fig. 7.3, this may lead to the execution of nonexistent topologic events, topology distortion and increase in numerical instability.

7.5.2 Redundant and Obsolete Events

Many of the computations of topologic events are redundant or obsolete. Some of the events will have to be computed more than once or will be computed but will never be processed, these events are called redundant or obsolete respectively. Consider the situation displayed in Fig. 7.4.

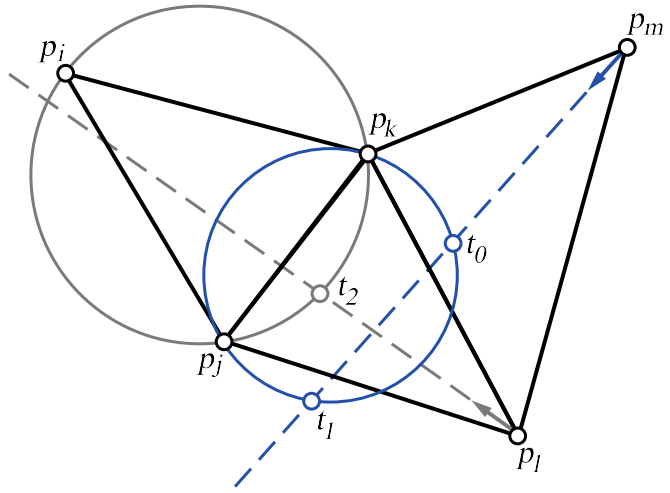
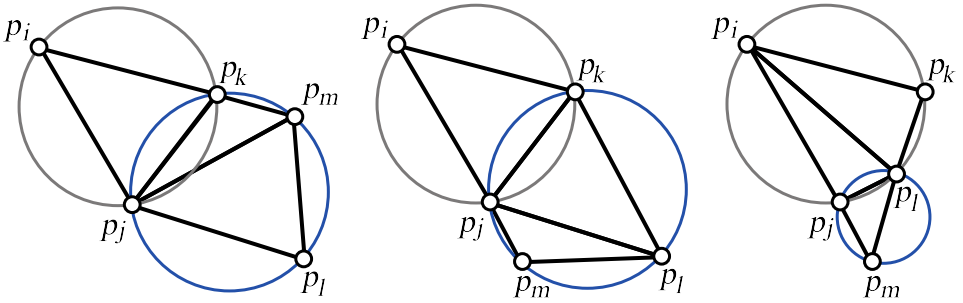


Figure 7.4: Redundant/obsolete topologic event.

A triangulation is shown in this figure, where a topologic event for triangles $p_j p_l p_k$ and $p_k p_l p_m$ will occur at time t_0 and another topologic event will occur for the same pair of triangles at time t_1. It is not necessary to spend time computing the later as only the first future topologic event for each triangle pair is pushed into the queue. Another topologic event is scheduled for triangles $p_i p_j p_k$ and $p_j p_l p_k$ and it will occur at time t_2. Events taking places at times t_0 and t_2 are computed with respect to the movement of the points p_m and p_l and 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 the points.



(a) Event scheduled for t_0 occurs. (b) The topology returns to original state. (c) Event scheduled for t_2 occurs.

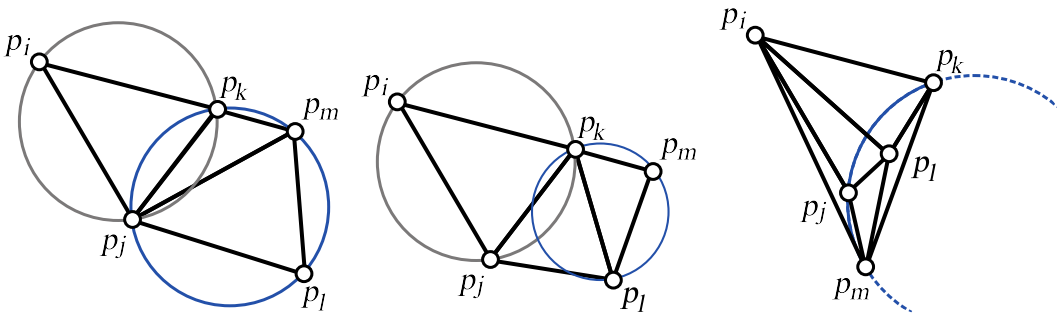
Figure 7.5: Topologic changes in the vicinity of an upcoming event ($t_0 < t_1 < t_2$).

Let $t_0 < t_1 < t_2$ (the events will occur in the following order: t_0, t_1, t_2). In this case the topology of the triangulation changes at t_0 as shown in Fig. 7.5(a), making the event scheduled for triangles $p_i p_j p_k$ and $p_j p_l p_k$ at t_2 illegal and removing it from the queue. After that, the topology changes again at t_1 as shown in Fig. 7.5(b), reverting topologically to the original state. This change invokes a new computation of topologic event for the triangle pair $p_i p_j p_k$ and $p_j p_l p_k$ which results in the topology event at time t_2 that has been already computed and discarded as shown in Fig. 7.5(c) thus making the event redundant. The situation can be even more complex when multiple points leave and enter the vicinity of a triangle pair similar to the one displayed in Fig. 7.4. In order to make the problem simpler to observe, the points in the figure are moved subsequently (point p_l remains static during the movement of p_m and vice versa). In the real application, the points would move simultaneously, but the speed of movement of p_l would be much slower than the speed of p_m.

Note: In the example shown in Fig. 7.5, there will be other events scheduled and executed between times t_0 and t_1. The example is simplified to provide a better insight into the nature of the obsolete events and to show that some of the events do not need to be computed if it is obvious that some of the preceding event will have changed the topology by the time of the obsolete event execution in such a manner that the event could not be processed.

In the case that $t_0 < t_2 < t_1$ (the order of the occurence of these events is that the event at t_2 will occur before the event scheduled at t_1), the event denoted by t_1 might not be executed at all and it is therefore obsolete. The situation is similar to the previous case - the first topologic event, scheduled at t_0 makes the event at t_1 obsolete and it is then removed from the queue. But due to the fact that $t_2 < t_1$, this exact topology event will never occur because the triangulation does not return to its original topologic state as shown in Fig. 7.6.

Similarly to the previous case, it can be seen that the first event that occurs at t_0 changes the topology of the triangulation as shown in Fig. 7.6(a). Then the event scheduled at t_2 is executed as shown in Fig. 7.6(b), but the topology of the triangulation at time t_1 is completely different. The triangles $p_k p_l p_m$ and $p_j p_l p_k$ do not exist at time t_1. Therefore, the Delaunay criterion is not violated and no event should be scheduled and executed for this time. The practical application of this principle will be discussed later in Chapter 8.



(a) Event scheduled for t_0 occurs. (b) Event scheduled for t_2 occurs. (c) Event scheduled for t_1 does not occur.

Figure 7.6: Topologic changes in the vicinity of an upcoming event ($t_0 < t_2 < t_1$).

Note: It is generally not practical to distinguish between redundant and obsolete events. The most important point of their detection is that their computation can be postponed until they are needed (if they are needed). Therefore, the terms can be used interchangeably in the context of practical application.

Part III

Applications of Kinetic Data Structures

Chapter 8

Hybrid Method for Managing Kinetic Delaunay Triangulation

Independentely of the research performed by Guibas and Karavelas in [38], we developed an advanced method that may be characterized as hybrid according to the terminology established earlier in this text (this classification only considers the method of the event times computation as we have already described in Chapter 7). It is important to note that *KDT* management includes several other parts which determine its overall functionality and performance. This chapter describes our implementation of the problem of kinetic Delaunay triangulation management (this research has been published in [92–94]).

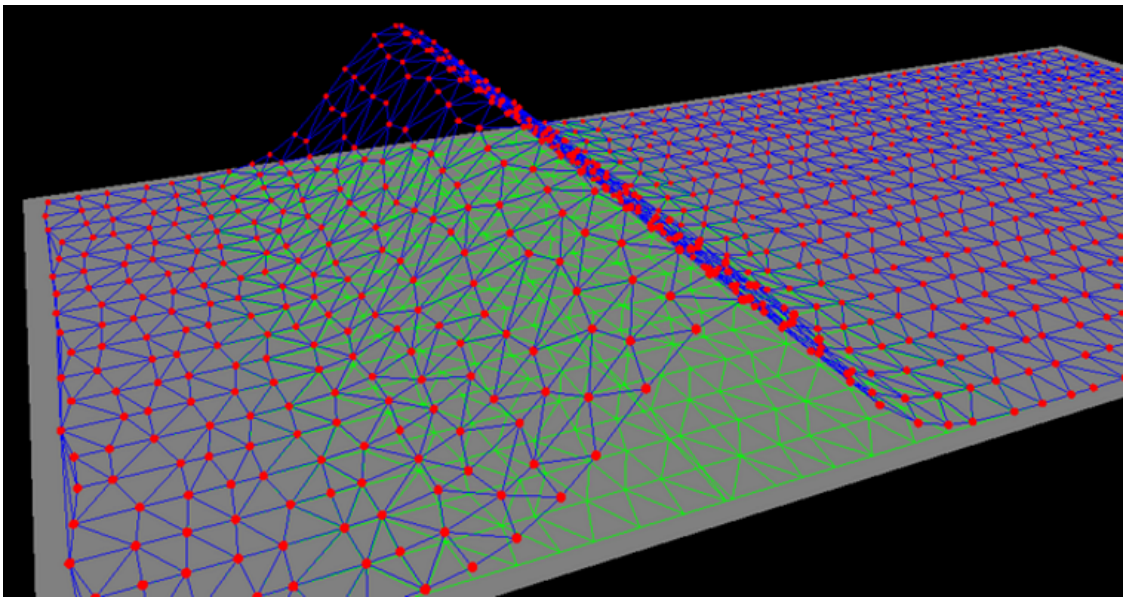


Figure 8.1: Experimental implementation of the kinetic Delaunay triangulation for 2.5D wave simulation.

For demonstration purposes, we have used our method to create 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. 8.1.

8.1 Basic Preliminaries

Construction Methods and Data Structures

From the variety of methods available for Delaunay triangulation construction, we decided to use the incremental insertion algorithm with a walk-based triangle search. This algorithm has several very important features with respect to the kinetization principle. At first, it is very simple to implement, it is online (we may simulate discrete time movement by removing and reinserting the points in the triangulation, if needed) and the search algorithm allows us to change the triangulation structure with relative ease. This last feature is especially important, because the movement of the points will almost surely cause a large number of alternations in the triangulation. If we used a DAG-based or similar lookup structure, we would have to perform relatively large-scale alternations in its structure, which would be prone to errors. From the variety of possible walking algorithms, we have chosen to use an orthogonal walk algorithm in combination with the Remembering stochastic walk (see [51, 81, 82]).

The online property also allows us to manipulate the dataset at any moment during the lifecycle of the application and (together with some point removal algorithm such as the one presented in [21]) makes the triangulation extremely versatile and even allows us to simulate the movement by the dynamic approach.

Point Movement

As stated earlier in the text, it is sufficient for most applications to limit the point movement to polynomial or even linear trajectories. For our implementation, we decided to limit the point movement to linear trajectories. Our experiments, as well as some of other applications, suggest that this type of movement, however simple, is usable for a wide variety of problems – e.g., collision detection applications may use a piecewise linear movement approximation and the motion interpolation problem also uses various kinds of linear movement functions. We will show how our own applications of *KDT* use linear movement of the generating points later in this chapter in order to simulate real-life scenarios.

8.2 Event Computation

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

In order to compute the events, we have developed a special method for polynomial root finding. This method is general and technically able to solve polynomials of any given degree, but since we only work with linear movement it is only used for polynomials of degree up to four in our context. The method is based on the information about the polynomial root locations and multiplicities that can be obtained from its Sturm sequence. The roots themselves are then found by combining this information with numerical methods.

Together with the fundamental theorem of algebra – [71], we may use the knowledge obtained from the Sturm sequence of a polynomial to create a table of guidelines for its solving. As said in Chapter 7, 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 8.1:

$\deg f(x)$	$f_m(x)$ **real root multiplicity**	$f(x)$ **real root multiplicity**
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	$\emptyset$ or {1, 1} or {1, 1, 1, 1}

Table 8.1: 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. And because only one root is left to recognize, this remaining root must be real, leaving $f(x)$ with only one possible root configuration - one double real root and one single real root, as shown in the second row of Table 8.1. Examples of all possible root configurations of a polynomial of the third degree without the corresponding Sturm sequences are shown in Table 8.2.

Using the aforementioned knowledge, we were able to formulate a hybrid algorithm for solving the polynomial equations. Its simplified form, Sturm_3 algorithm, is shown in Alg. 4. This simplified form describes the approach we use for solving polynomials of the third degree, solving polynomials of higher degrees uses the exact same principle, but the number of possible polynomial root configurations grows with increasing degree of the polynomial being solved.

The Sturm sequence of the given polynomial $c(t)$ (i.e., the certificate function) is constructed and used to obtain two important pieces of information – the total amount of roots and all the multiple roots of this polynomial (as we stated in Chapter 4, both the exact locations and the multiplicities of the multiple roots may be obtained in this fashion). If any multiple roots exist, we divide $c(t)$ by the polynomial $(t-t_1)^{r_1} \cdot (t-t_2)^{r_2} \cdot \ldots \cdot (t-t_n)^{r_n}$ where $t_1, t_2, \ldots, t_n$ are the multiple roots with mulitplicities $r_1, r_2, \ldots, r_n$. If $c(t)$ does not have any multiple roots, it is left unmodified. Together with the fundamental theorem of algebra, we are now able to solve $c(t)$ by using either the analytical formulas if the degree of the resulting polynomial is equal to one or

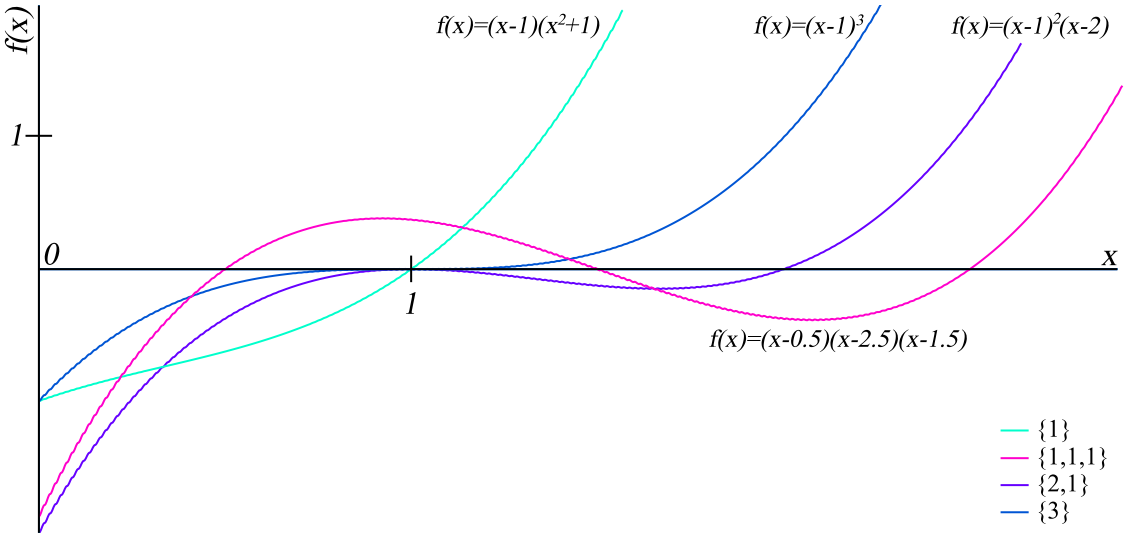


Figure 8.2: Examples of all possible root configurations of a polynomial of the third degree.

two, or suitable numerical methods if the degree of $c(t)$ is greater. For the numeric part of the approach, we use the method of bisection for the initial root position estimation and Newton's method to enhance the precision of this estimation to the required value. We have chosen these methods because of their simplicity in the case of bisection and because of their extremely easy implementation and excellent expected performance in the case of Newton's method. We have also tried solving the lower degree polynomials numerically, but the results were no more precise than those obtained by the analytical approach and the analytical approach is very easy to use and fast since the analytical solutions are well known for the linear and quadratic functions.

Note: The $Sturm_3$ algorithm, as described in Alg. 4, contains a check for empty set of roots (see lines 3–6). According to the fundamental theorem of algebra, it is not possible for a polynomial of non-even degree to have zero real roots. This part of the algorithm is mentioned solely for illustrational purposes here and will be used when solving polynomials of even degree.

Algorithm 4: Sturm$_3$ Algorithm.

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

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

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

$f_1(t), ..., f_m(t) \leftarrow$ Sturm sequence of $c(t)$
$r \leftarrow (V(-\infty) - V(\infty))$
if $r = 0$ **then**
  Return empty set of roots
end
$R_m \leftarrow$ set of r_{mult} roots of $f_m(t)$
if $\|R_m\| = 2$ **then**
  Return $\{r_{m1}, r_{m1}, r_{m1}\}$ `// One triple root`
else if $\|R_m\| = 1$ **then**
  $r_s \leftarrow$ the only single root of $\frac{c(t)}{(t-r_{m1})^2} = 0$
  Return $\{r_{m1}, r_{m1}, r_s\}$ `// A double and a single root`
else
  Return $\{r_1, r_2, r_3\}$ `// Set of three distinctive roots`
end

8.3 Redundant Event Reduction

As shown in Chapter 7, some of the topologic events that are computed will be descheduled before their execution and the runtime spent on their computation is wasted – we call these events redundant. Our research shows that the redundant events represent roughly 50% of all the events that are computed during the application runtime as shown in the graph in Fig. 8.3.

The graph in Fig. 8.3 shows the total amount of scheduled topologic events for various percentages of moving points in the generator set. The test was concluded on a set of 100 points that were randomly placed in a square-shaped area and a certain subset of these points was assigned a random velocity vectors. After ten seconds of *KDT* management, we were able to determine that the total number of succesfully processed topologic events is nearly equal to the number of the discarded events which means that approximately half of the time spent on the event computation is lost. Furthermore, as we can see in the graph in Fig. 8.4, the runtime spent on the event computation represents over 90% of the total time needed for the *KDT* manegement if at least 40% of the points are moving.

As shown in the previous text, topologic event computed for two adjacent triangles at time t_e will become redundant if at least one of the two triangles is removed from the triangulation at any time $t_r < t_e$ (such a removal will usually occur as a result of an edge swap). With a relatively little effort, this knowledge may be used to reduce the number of computed events by assigning

a time of the next event scheduled for that triangle to each triangle in *KDT* or an infinite value if no event is scheduled for the triangle as illustrated in Fig. 8.5.

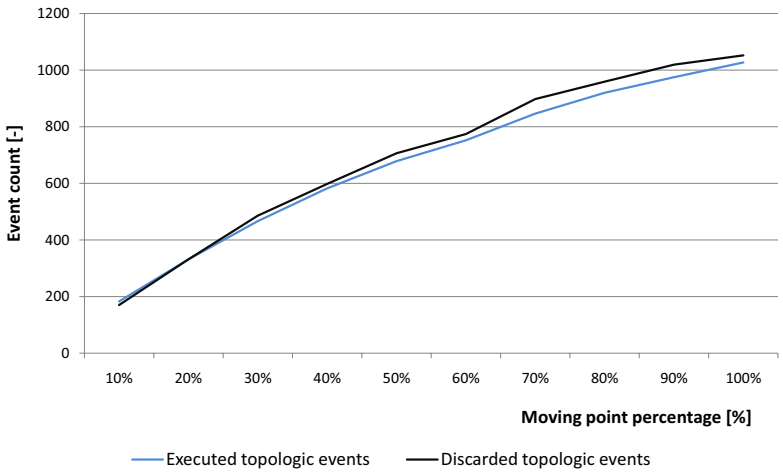


Figure 8.3: Total amount of scheduled and discarded topologic events.

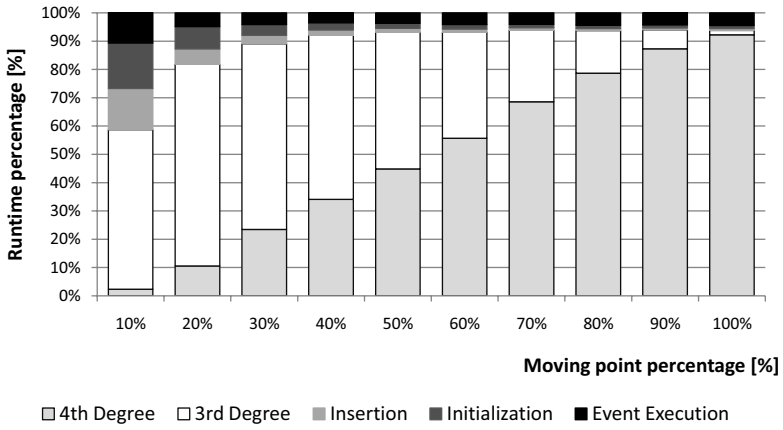
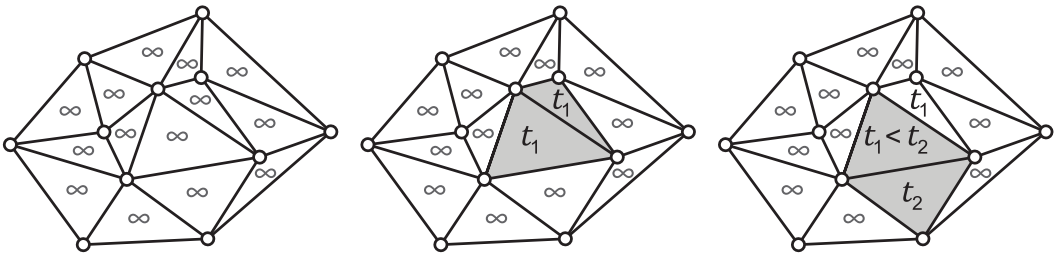


Figure 8.4: Distribution of runtime consumption during *KDT* management.

Let us first consider the initialization step as shown in Alg. 3 (presented in Chapter 4). At the beginning of this step, we have a Delaunay triangulation and no computed topologic events. We now assign an infinite value to each triangle as a time of the next topologic event that will alter this triangle – see Fig. 8.5(a). Following the aforementioned algorithm, we now choose pairs of adjacent triangles and try to compute the first topologic event that will occur for the two chosen triangles. If such an event exists and will occur at the time t_1, we push it into the priority queue and assign the value t_1 to these two triangles as shown in Fig. 8.5(b) (the triangles are marked by grey color in the figure). Later on, we may try to compute a topologic event for a triangle pair which contains a triangle with a non-infinite time value as shown in Fig. 8.5(c). Let us say such an event exists and will occur at the time t_2. We will then assign the value of $\min\{t_1, t_2\}$ to each of the two triangles in the currently handled triangle pair, where t_1 is the original time value assigned to the triangle. In this case, the redundant topologic event is not computed and it is thus necessary to combine this approach with such a computation method which allows us to compute the topologic events separately for given time intervals.

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



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

Figure 8.5: Initialization step with redundant event reduction.

The graph shown in Fig. 8.6 demonstrates the changes in the number of computed redundant topologic events if we use this method. As we can see, the number of redundant topologic events drops approximately to one half of the original number. We may see that only one third of the computed topologic events is now being discarded as redundant events (in other words, we are able to detect approximately 30% of the redundant events before they are computed).

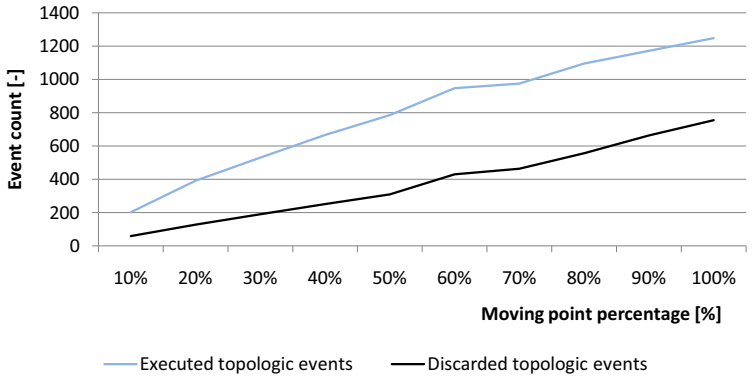


Figure 8.6: Total amount of scheduled and discarded topologic events when using redundant event reduction.

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

Chapter 9

Video Representation based on Kinetic Delaunay Triangulation

As a part of our research, together with *Petr Puncman*, we have developed a method for video representation based on kinetic Delaunay triangulations. The triangulation is constructed over a collection of samples from selected intra-coded frames, point movement vectors are determined by precomputed optical flow vectors gained by block matching algorithm in inter-coded frames and decoding data from samples by means of barycentric interpolation and feature based warping. The results of our research have been been published in [96].

9.1 Introduction

The way of the digital video representation plays a crucial role when determining the intended quality, compression ratio or even the target application of such a video record. The most recent techniques in this area often use not only a reduction of the spatial redundancy but they also try to reduce the temporal one by using a wide variety of techniques, including both the loss and lossless compression of the video frames, conversion to the frequency domain and back and others. The frames of a video record may be seen as a set of unconnected images or the similarity of the consecutive frames may be exploited by some methods. Our work investigates a possibility of using the kinetic data structures, namely the kinetic Delaunay triangulation, in connection with a method for a spatial redundancy reduction. These structures retain some of their special properties despite the point movement and thus enable us, when used together with techniques for motion compensation, to track relevant points in order to compensate differences between consequent frames and thus maximize quality at low bitrates.

9.2 State of the Art

9.2.1 Video Compression

The most widely used present algorithms for lossy video compression concentrate either on the intra-coded frame modifications in frequency domain such as the discrete cosine transform (MPEG1-2, DV, MJPEG, H261-4) and the discrete wavelet transform (MJPEG 2000, Intel Indeo

5) or on the vector quantization (Cinepak, Sorenson Video). Inter-coding, if there is any, is often handled by the block matching algorithm (*BMA*). It presents the simplest way of obtaining the motion vectors of the corresponding macroblocks (16×16 pixels group of four 8×8 pixel blocks - a basic element for video compression) over the whole frame (e.g., in Fig. 9.1 on the right side we can see a block B which has been found to be similar to the block $\hat{B}$ in the same window in the next frame). The resulting video is then partitioned into picture groups containing intra-coded I-frames, inter-coded P-frames and optional bidirectionally inter-coded B-frames retaining their mutual relations (see Fig. 9.1 on the left).

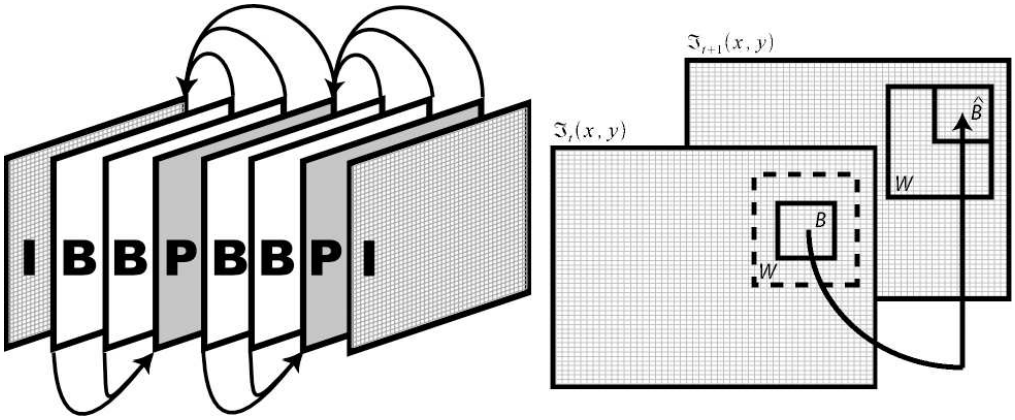


Figure 9.1: Typical compression scheme (left). Block matching principle (right).

The principle of the abovementioned intra coding is that each frame is considered to be a standalone image and is processed accordingly. Thus each I-frame is encoded without using any information from the previous (or the following) frames in the video sequence. On the opposite, the inter-coded and bidirectionaly inter-coded P-frames and B-frames use their similarity to the surrounding frames (in one or both directions) for the encoding. In most cases, simply put, *BMA* searches for such movement vectors for each macroblock that, if applied to the macroblock in question, describes the following frame with minimal error.

The most common problems connected with classic video compresion schemes are the appearance of new block elements between two frames and a serious loss of detail at low bitrates. Also handling these video representations is a bit impractical, when there is a need to transform or interpolate a videosequence. The corresponding frames have to be decompressed and all their pixels processed. These drawbacks may be solved, in our case, by using an alternative triangulation-based video representation. Present solutions on this topic deal with the movement of 2D and 3D triangulations that represent synthetic objects in the video (see [1, 103]). Other approaches include the construction and movement of adaptive triangulations over the whole scene (see [83, 102]). The last group of approaches considers the videosequence to be a 3D object which is then tetrahedronised (see [89]).

9.3 Video Processing by Kinetic Delaunay Triangulation

In our approach, we focus on the combination of the methods mentioned in [1, 103] and [83, 102] together with the abilities of *KDT*. That means that we perform both sampling of the pixels of the original image in coder and interpolation or warping of the full scene by using *KDT*. The idea of video representation by *KDT* is based on a creation and successive movement of a triangulation.

In the first step, the relevant points and some random points have to be obtained from a frame which is considered to be intra-coded. These points then define the *KDT* until a new picture group is formed, starting with the next intra-coded frame. The second step means processing frames between the two following I-frames to form inter-coded frames by moving the points in the triangulation and reconstructing the frames from the current state of the triangulation. The movement of the points in *KDT* is defined by the vectors obtained during motion estimation. So the whole process may be described as follows (components of this process will be described in the following two sections):

Important point selection: A set of important points is selected from the input image. *KDT* is created from these points.

Motion estimation: Corresponding blocks centered around each vertex are found in consecutive frames. Their positions are then used to define motion vectors.

Movement of *KDT*: The movement is initialized transforming the current frame into the following one.

Topologic events: Topologic events are computed and handled.

Stability improvements: Means for improving stability are introduced.

Video decoding: A frame is reconstructed from the data stored in the *KDT*.

The use of *DT* is crucial for our method if we want to obtain usable compression ratio. It is vital to note that for every other type of triangulation we have to store not only the coordinates and color of each vertex but also an information on the surrounding topology. On the other hand in *DT* we may be sure that if no four points in the triangulation are cocircular, we will always reconstruct the same triangulation (independently of the algorithm used) without storing any additional topology data. As we have shown (see [68]) the compression may be expected to be meaningful (considering a method without any stored topology information) with up to 20% inserted points if we use a trivial compression method and up to 30% inserted points if a more sophisticated method, which exploits delta coding, is used.

9.3.1 Selecting the Points of Interest

To obtain valuable samples from the input frame, we mix edge points with random points at the ratio of 1:1 (the best ratio value was obtained from the performed ratio tests). Starting with an already given point count, we select the most suitable points from the frame by the methods of discrete convolution, selective thresholding and randomized selection. The resulting set of pixels then in the ideal case contains a predefined amount of points. These points include dense pixels on the edges, sparse pixels on the homogenous surfaces and uniformly distributed pixels everywhere else. For details on selecting the points see [68] as it is beyond the scope of this text. Finally, *KDT* is created by incremental point insertion from the selected points.

9.3.2 Motion Estimation

Although the block matching algorithm often deals with macroblocks covering the whole frame, in our case we had to find which block in the next frame makes the best match for the block

centered around each *KDT* point in the current frame. Each examined point then grows into a suitable search region W (see Fig. 9.1) which is intensively inspected. For an evaluation of the differences between the blocks B and $\hat{B}$ we have used three different metrics: Mean Square Difference (*MSD*), Mean Absolute Difference (*MAD*) and Pel Difference Classification (*PDC*) - see [30]. These metrics allow us to recognize the corresponding blocks and thus obtain the needed motion vectors. Once all the vectors in a frame are known, velocities of their points in *KDT* are set (see Fig. 9.2).

9.3.3 Video Decoding

The previous steps provided us with the most important points in each frame and their inter-frame correspondency thus allowing us to compute the motion vectors for these points. By inserting the points obtained in each key frame into a *KDT* and moving them along the motion vectors we get a sequence of triangulation states that represent each frame between two keyframes. From these triangulations we must now decode the approximation of the original frame.

Figure 9.2: From the left: the original *KDT*, motion vectors, compensation of *KDT*.

In the case of the intra-coded frames, we have to use an interpolation algorithm. At first we have to obtain a raster representation of the edges e_1, e_2, e_3 belonging to each triangle $p_j p_k p_l$ (see Fig. 9.3). For this purpose we use an implementation of Bresenham algorithm [13].

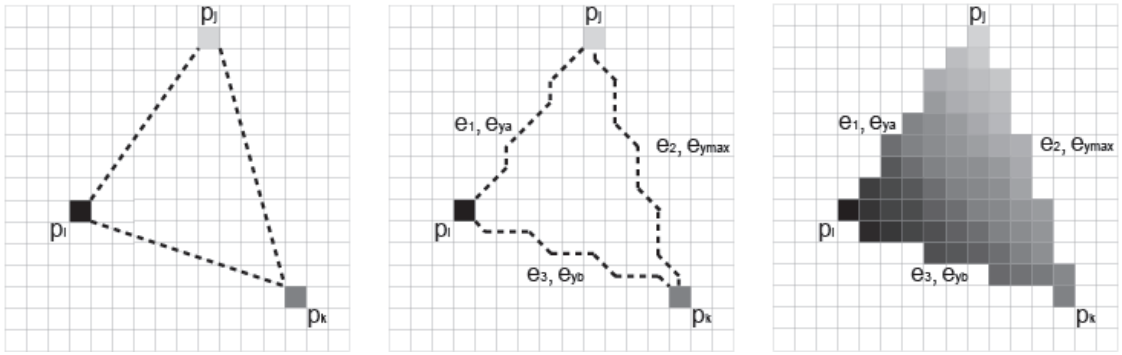


Figure 9.3: (a) vertices with associated intensities, (b) edges by Bresenham algorithm, (c) scan-line fill and triangle interpolation, [13].

Successive computation consists of (among other equations) solving Eq. (9.1) for all three triangle vertices. By solving this system of equations, we obtain the coefficients a, b, c which are then used to compute the intensities of general points inside the triangle.

$$\begin{pmatrix} x_i & y_i & 1 \\ x_j & y_j & 1 \\ x_k & y_k & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} z_i \\ z_j \\ z_k \end{pmatrix} \tag{9.1}$$

where $[x_i, y_i], [x_j, y_j], [x_k, y_k]$ are the coordinates of the vertices of a triangle and z_i, z_j, z_k are the intensities of the corresponding pixels.

In the case of inter-coded frames, we move *KDT* as described in Section 9.3. After then, either the aforementioned interpolation or feature based warping (see Fig. 9.4) may be applied.

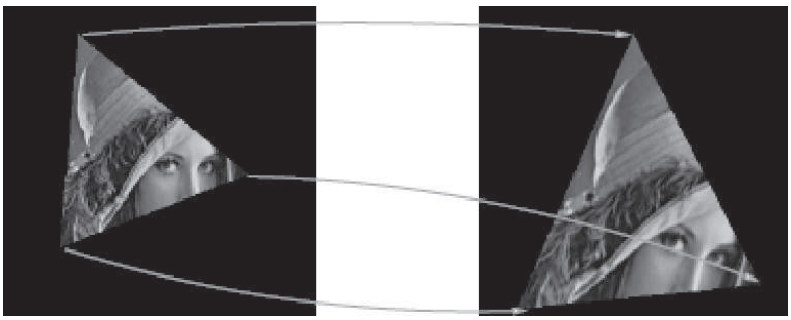


Figure 9.4: Triangle warping, [57].

In our case, the warping process takes the edges of a triangle in the current frame and corresponding transformed edges in the previous frame. The triangle can change significantly from frame to frame and the task is to compute the intensities of the pixels within the transformed triangle. As we have a relatively accurate approximation of the intensities of all the pixels in the last intra coded frame, we perform the warping process for all the consecutive inter-coded frames after that frame. An incremental computation is also possible but it leads to notable loss of accuracy. We adopted the warping process for more line pairs as described in [69] gaining X' original pixel coordinates. The positions of the three points X'_i are then passed to weight function (9.2) with the weights w_i being proportional to the pixel-edge distance in order to produce more accurate pixel position X.

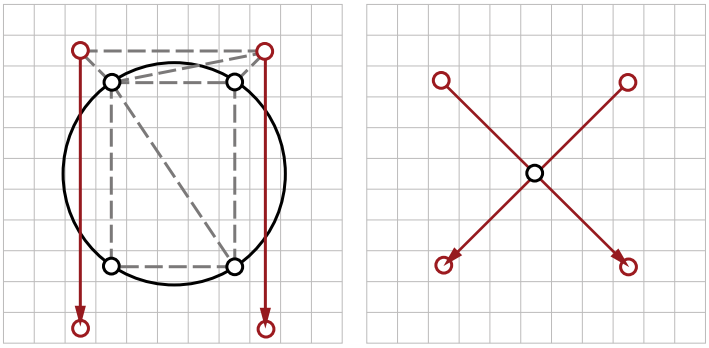
$$X = \frac{\sum_{i=1}^{n} w_i X'_i}{\sum_{i=1}^{n} w_i} \tag{9.2}$$

9.4 Managing the Kinetic Delaunay Triangulation

In our experiments we use the kinetic Delaunay triangulation as described in the previous text, but the specific environment of rasterized image data imposes very special qualities on the triangulated data. Because of the fact that the triangulated points exist within a regular grid and it is quite common for multiple points to be cocirular within the grid, the triangulation is bound to become locally singular very often. These local singularities represent a potential numerical stability issue during the computation of kinetic events and have to be addressed.

KDT Stability Improvements

Due to the fact that all the points inserted into the triangulation are representation of pixels in a grid and their velocities reflect their movement in this grid, we may easily encounter unwanted singular cases which become the source of various stability issues. These singularities may include for instance collisions of two moving points or number of points becoming cocircular at the same moment. Illustration of these singularities is given in Fig. 9.5.



(a) Six cocircular points. (b) Point collision.

Figure 9.5: Singular cases for grid point movement.

Our experiments show that there are two reasonably simple ways of removing these singularities. We may either add a small random number to the velocities or to the initial coordinates of the points (or even to both of them). Only the moving points or all of the points in the triangulation may be altered in this fashion. These numbers must be of course small enough to be later eliminated by the rounding process without moving the points into wrong cells in the grid. We have tested both the mentioned types of randomization and various random values and finally we managed to determine the ideal combination - we used only velocity vector components randomization with uniform distribution of the random part and 10^{-3} maximum absolute value of the random addition. This effectively prevents the points from colliding as well as becoming cocircular in large numbers.

9.5 Experiments and Results

The test application was implemented in C# and its purpose was to provide the results we needed for algorithm efficiency evaluation and for comparison with existing solutions. We wanted to test the main properties of our algorithm, such as the quality of the video which is indicated by *PSNR*, bitrate in bytes per frame and the overall performance of each step.

The tests were performed on three videosequences with the length of 100 frames, resolution of 176×144px and the length of the picture group set to 6. The sequence with the dynamic camera contains a lot of synchronized movement, the synthetic sequence contains a very large percentage of untextured and simplified surfaces and the talking head is mainly static. The quality criterion is controlled by the *PSNR* metric - see Eqn. (9.4).

$$MSE = \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \|\Im(i,j) - \Im'(i,j)\|^2 \tag{9.3}$$

$$PSNR = 10 \log_{10} \left(\frac{255^2}{MSE} \right) \tag{9.4}$$

where $M \times N$ is the dimension of the image, $\Im$ is the original image and $\Im'$ is the decoded approximation of the original image.

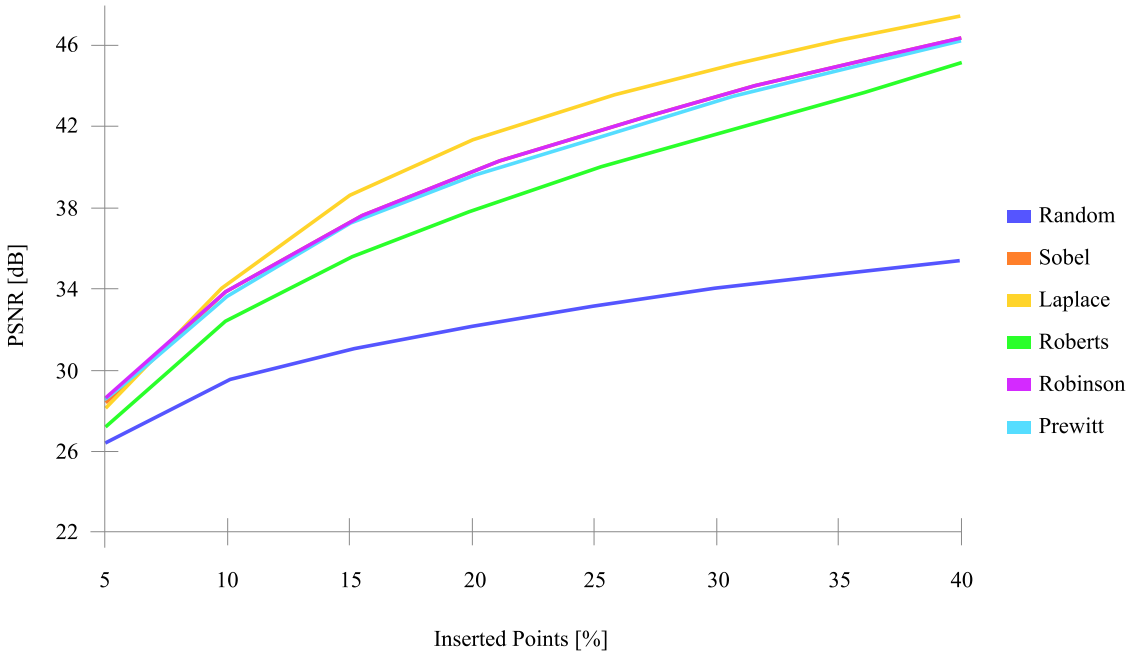


Figure 9.6: Dependency of *PSNR* on inserted points count.

In order to examine the quality of intra coding we studied an influence of mixed (50:50 edge:random) point selection against to solely random point selection. The set of chosen edge operators (random, Sobel, Laplace, Roberts, Robinson and Prewitt - for detailed information on these operators, see [50]) - see Fig. 9.6 - provides noticably better results than a random generator at the whole range of inserted points percentage. Note that acceptable *PSNR* value starts at 30dB.

The second comparison we made (see Fig. 9.7) shows the bitrate achieved in each frame of all the three test sequences. The size of an inter-coded frame is derived from the amount of the motion in the scene. At these bitrates we were able to provide compression ratio of around 20:1.

In the third test (see Fig. 9.8), we compared the quality of our solution with XviD for the same coded output size. The initial amount of inserted points was 5% and the length of the pictures group was set to 6. The intra-coded frames provided quality nearly as good as XviD did, but inside the inter-frames quality dropped rapidly. Both techniques of decoding (interpolation and warping) were measured to be nearly equivalent, however, subjective comparison often prefers the warping prior to the interpolation because of triangular artifacts which may appear as a result of the interpolation.

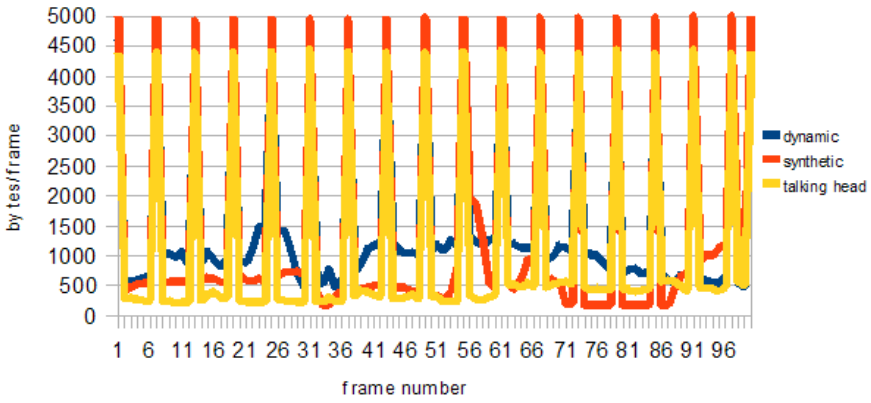


Figure 9.7: A Detailed Bitrate Behavior of Intra + Inter Coding for 5% Points.

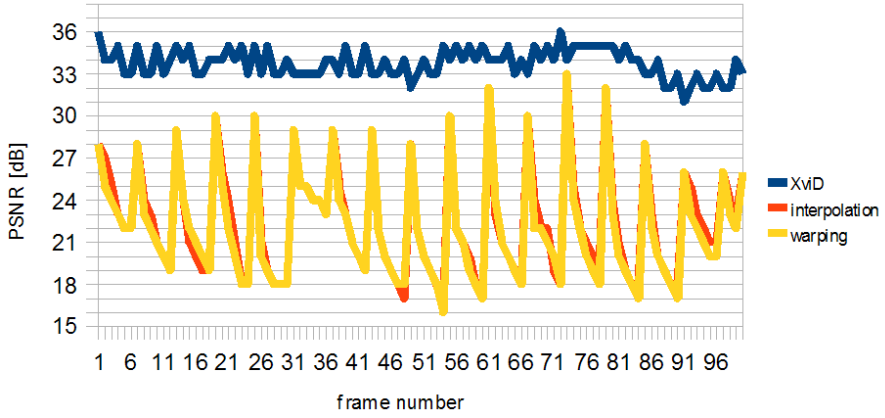


Figure 9.8: Intra + Inter Coding vs. XviD Quality Comparison.

The tests showed that if we use intra coding for the video frames only, all the used operators behave in a similar fashion and their quality response is in all the measured cases logarithmic (measured in *PSNR*). However, the best results were received for video sequences that contained large percentage of homogenous areas, such as rendered videos and talking head videos, because these types of video sequences are very suitable for coding and motion blur occurs very rarely, if at all.

9.6 Conclusion

Various requirements are often set for a new video representation method. These may include the ability to perform morphologic changes on the triangulation and receive the corresponding changes in the image matrix of the video, minimization of the bitrate and maximization of the quality. With these possible requirements in mind, we have designed and implemented a new method that allows us to encode the frames into a geometric form according to the primarily requested feature.

Inter-coding showed its major advantage in very low bitrates and good compression ratio. For 5 - 30% inserted points, we were able to achieve compression ratio from 20:1 to 4.5:1. Detailed tests and their results may be found in [68].

From the point of view of the kinetic Delaunay triangulation, the most important drawbacks (the lack of stability and the occurence of singular cases) of the application were succesfully removed by introducing the randomization of the velocity vectors of the moving points. However some performance issues are still left to be solved, especially in the area of computing the topological events, of which is a large percentage computed but not executed. Introducing some kind of nonlinear movement may also pose some advantage in this kind of application (for instance the movement along elliptic trajectories).

Chapter 10

Early Warning System for Air Traffic Control

In cooperation with *CS Soft Group* [17] we have explored the possibility of using kinetic Delaunay triangulation as a base for an early warning system to be used by the air traffic control. The problem basically represents a collision detection with the feature that the upcoming collisions have to be detected with sufficient advance so the aircrafts may take some kind of evasive maneuver to leave the collision course. The application is similar to the one presented in [35] for marine environment but poses several key differences which we used to test several theoretical expectations of the *KDT* behavior. The research has been published in [90].

10.1 Introduction

In the field of air traffic control it is vitaly important to be able to detect potentially dangerous situations in such an advance that the upcoming danger can be avoided. Currently the air traffic is monitored and controlled mostly by human operators. Each of these operators obtains a certain small subset of the air traffic data provided by various tracking devices such as radars, etc. Our approach provides a global perspective of the problem. By analyzing the whole set of aircrafts detected by the radar at once, we are able to detect potential threats before any other postprocessing is applied to this set and to detect such situations when two aircrafts are on a crash course.

In order to detect these cases, we exploit the *KDT* as described in the previous text. Among the various features of the *DT*, we use namely the fact that all the points in *DT* are connected with their nearest neighbors. Combined with the movement of the points, it provides us with an efficient collision detection system, see [31]. The *KDT* was already used by Gold and Condal for similar purpose in a marine environment, see [33]. However, the use in the aeronautical applications differs in some key features. Furthermore, during the implementation of our application we tested several theoretical expectations of the *KDT* behavior.

Even though our method represents a global view on the problem and thus is not suitable for use in the current air traffic control, it shows a possible way of development in this field. Should the currently used approach be replaced by some kind of a global control system, our application would represent one of the candidates for the new system.

10.2 Air Traffic Control Systems

10.2.1 Current Methods and Conventions

According to [17], the operators who control the movement of the aircrafts in certain portion of the air space are each assigned only a small portion of the nearby traffic. Thus each human operator is responsible only for approximately ten aircrafts. When an aircraft lands or leaves the space assigned to a certain operator, that operator's control over this aircraft is lost or transferred to some other operator respectively.

Furthermore, the aircrafts may (except some special cases) only move through strictly defined corridors called routes, which are defined differently for different kinds of aircrafts (e.g., helicopter routes may differ from international flight routes). The routes also differ in accessibility – certain routes are only available for some heights, specific visibility conditions, etc.

The predefined heights on which the aircrafts travel along the routes are called flight levels. The flight levels, even though they are defined precisely, vary with changing atmospheric pressure. This effect is caused by the definition of the flight levels. The flight level is a level of constant atmospheric pressure which causes the aircrafts to flight in different heights for different atmospheric conditions even though the aircrafts are located on the same flight level. Further information on the routes, flight levels and other air traffic rules may be obtained in [4].

10.2.2 Air Traffic Data Sets

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

The data in the set are only sorted timewise – the records are logged as the radar detects them. Each record contains the location, height, heading and velocity information about one plane. Note that the data set only contains information about the *current* whereabouts of a given aircraft. We do not know anything about the overall aircraft trajectory, whether it is currently flying forward or is in the middle of some kind of aerial maneuver, etc. This fact is a consequence of the means of obtaining the data and although it may seem restricting, it correlates very well with the data structure we use, because our implementation of *KDT* only allows linear trajectories of the moving points.

10.3 Geometric Features of the Problem

10.3.1 Problem Overview

Let us briefly summarize how we handle the given problem with our application. As we may see in Fig. 10.1, the application may be divided into four steps. In the first step the radar provides us with aircraft data; these data are then projected into the Euclidean plane (step 2), a kinetic Delaunay triangulation is constructed over the projection (step 3) and used for detecting possible collisions (step 4). The following text describes steps 2 and 3.

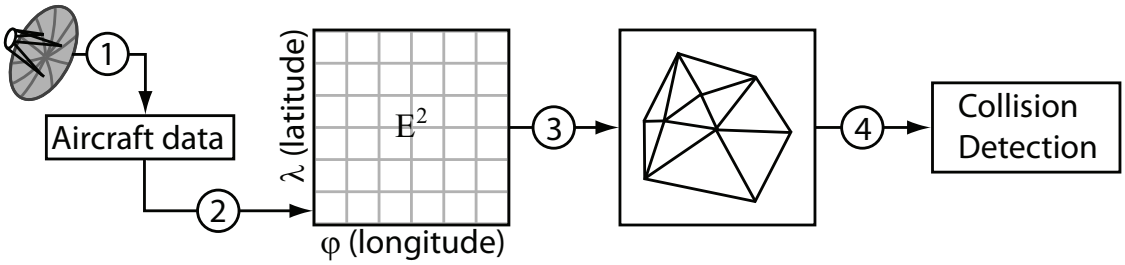


Figure 10.1: Block scheme of early warning system for air traffic control.

10.3.2 Aircraft Movement Mapping to 2D

Due to the restrictions forced upon the aircraft movement (the flight levels and routes), the problem of aircraft navigation is in fact (simply put) only a 2.5D problem. As such, it may be managed by exploiting a planar triangulation with height values assigned to each of the planes. The triangulation will only use the latitude and longitude coordinates of the planes to check for collisions. If a collision is then detected in the 2D projection, the relevant planes will be checked in full 3D and the collision warning will be proposed further only if the collision will really occur.

For the purposes of our application, we used a simple planar projection using *WGS84* model of the Earth for the coordinate transformation (see [62]). *WGS84* is the most commonly used model in the field of air traffic control. Another problem related with the aircraft mapping to 2D is that the aircrafts do not fly along linear trajectories (especially when waiting above an airport, the aircraft may be flying in circles for a prolonged period of time). Using the newest available data from the radar readings for each aircraft and one of the mentioned Earth models, we predict the next position of the aircrafts by using a simple linear extrapolation. Fig. 10.2 shows the difference between the actual path of an aircraft (solid black line, denoted p_{act}) – with marked positions when the aircraft is located by the radar – and the path predicted using the linear extrapolation (dashed gray line and filled gray circles, denoted p_{pred}).

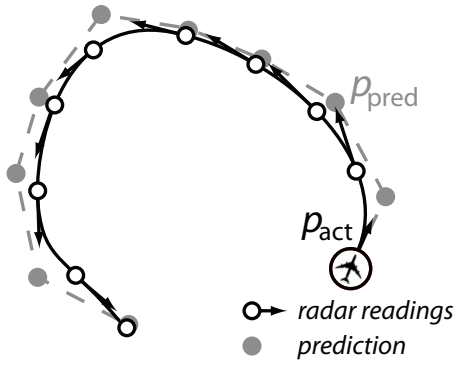


Figure 10.2: The difference between the real path of an aircraft (black line, empty circles) and the extrapolated path with some position tolerance (gray dashed line, filled circles).

10.3.3 Kinetic Delaunay Triangulation Modifications

Besides the topologic events, the topology of the triangulation will also need to be changed when a new information about an aircraft is obtained from the tracker. This will happen every time when the newly obtained position $p_q(t+\Delta t)$ differs from the position precomputed by extrapolation $p_q(t)+\mathbf{v}_q\cdot\Delta t$ (see Fig. 10.2) by more than a given tolerance $\tau \geq 0$ as shown in Eq. (10.1).

$$\frac{\|p_q(t+\Delta t)-(p_q(t)+\mathbf{v}_q\cdot\Delta t)\|}{\|p_q(t+\Delta t)\|} \leq \tau \tag{10.1}$$

If the condition in Eq. (10.1) is satisfied, we modify the point $p_q(t+\Delta t)$ so that its velocity vector remains unchanged (i.e., we use the newest information obtained from the radar at t_{i+1}) and its position is set to $p_q(t)+\mathbf{v}_q\cdot\Delta t$ (i.e., the position predicted at time t_i). The error generated by this approach may increase with each iteration but it will be corrected as soon as it reaches the value given by τ. To do so, we remove the point from the triangulation by using a removal algorithm similar to, e.g., [21] and reinsert it back on the correct position.

The fact that the information about each plane gets periodically updated may be used together with the polynomial solving method mentioned in the earlier chapters. Since we expect the information to be slightly incorrect by the time the update is received and the velocity or position of the plane will need to be updated, we may limit the computation of future events by the value of time of the next update on the aircraft position.

10.4 Results

As said before, the triangulation behavior will be different if we allow some small inaccuracy τ of the position of the aircrafts. Our experiments show that τ should be several percent at most – this will allow us to only update the velocity vectors of the aircrafts as new radar readings are obtained by the application thus increasing the performance of the application as shown in Fig. 10.3.

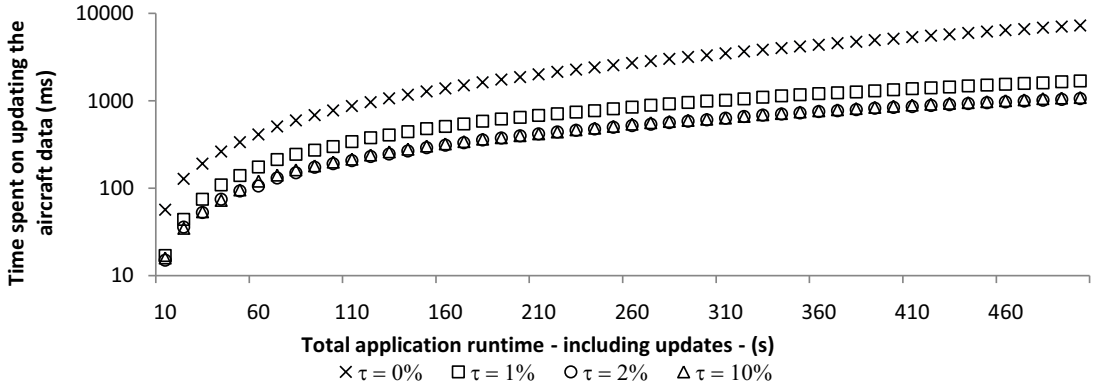


Figure 10.3: Comparison of the performance for various values of τ.

We may see that introducing $\tau > 0$ reduces the time consumed by updating the aircraft positions to about 15% of the time for $\tau = 0$. This will become important for larger datasets (e.g., thousands of planes).

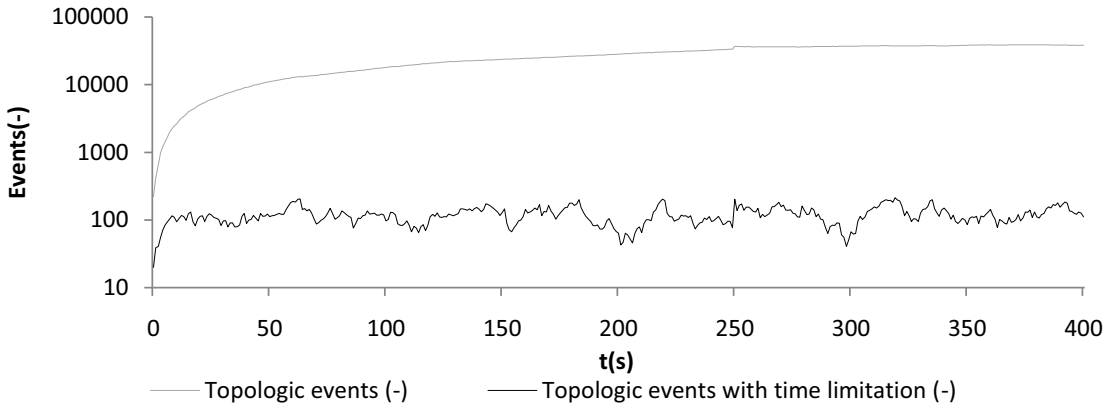


Figure 10.4: Length of the priority queue with and without the time threshold.

Due to the fact that the radar antena is rotating, we may omit those events which will happen after one rotation because by then, we will have obtained new aircraft data and use them to compute new events. The graph in Fig. 10.4 shows us the difference in queue lengths with the thresholding on the event computation on and off – we may clearly see that the event time thresholding reduces the length of the priority queue to approximately 1% of its original size.

10.5 Conclusion

In the given scale of the air traffic over the Czech Republic, our algorithm is stable and very fast. If used for larger datasets (more intense air traffic) we anticipate it to be usable even for situations where the number of the currently monitored aircrafts increases dramatically. Our method shows a possible way of future enhancement in the field of air traffic control as it provides a global point of view and is able to detect potentially dangerous situation automatically.

Chapter 11

Corridor Selection for Virtual Pedestrian Navigation

As mentioned earlier in Chapter 5, the agent-based simulation methods manage the crowd by navigating each pedestrian individually which is a task that (generally speaking) consists of two sub-tasks. First, a coarse path (also called a corridor) of each individual pedestrian has to be found, thus obtaining the general route that pedestrian will take in order to reach the desired destination from its current position. Second, the pedestrians are iterated through the precomputed corridors using a local navigation method such as a social-force model as needed by the simulation. As a part of our research with *Jakub Szkandera*, we explored and compared several ways of obtaining the corridors with the aim to obtain as natural corridor for each given pedestrian as possible. The result of our work was an unpublished manuscript, currently available online, see [97].

11.1 State of the Art

The schema of the whole corridor search process is shown in Fig. 11.1. The process consists of several steps which may differ slightly depending on the actual algorithm used. On the input, there are two main sets of data – the environment and the pedestrians. The environment part of the input is a composition of the *environment geometry* and *areas of preference*. The environment geometry describes the simulation area itself, e.g., the terrain relief, boundaries of the area and the shape of the unmoving obstacles. The areas of preference are used for further customization by describing which areas are prefered by which pedestrians. The *pedestrian data* provide the pedestrian information such as their starting locations and destinations. These input data are then used by the corridor searching algorithm which provides the pedestrian corridors on its output.

As far as the methods of corridor searching are concerned, the problem of finding the path leading from each agent's starting position towards its destination is principially equal to the problem of finding a path in an undirected weighted graph. Therefore, either heuristic methods, such as variations of the $A*$ algorithm [41, 55] or precise methods, such as Dijkstra's path searching [85], are most commonly used.

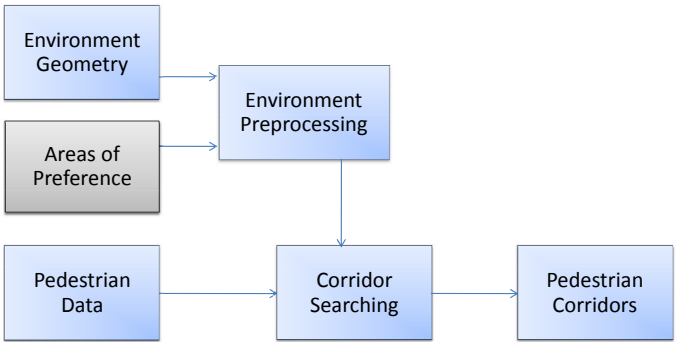


Figure 11.1: Schema of the corridor searching.

Grid-Based Spatial Subdivision

Perhaps the most straightforward method of environment representation is grid-based spatial division, especially in the field of computer games. However, in the context of corridor searching, this type of environment representation is not very common since its properties are best used by different types of crowd simulation that does not use corridors at all [15, 77, 87].

Waypoint Maps and Corridor Maps

Waypoint map (sometimes also called roadmaps) is a graph which describes the accessibility of different parts of the environment by the virtual pedestrians. Its vertices are called *waypoints* and represent special locations in the environment such as road crossings and the edges connect those waypoints that are mutualy reachable. A corridor map represents a system of collision-free corridors for the static obstacles in a given environment [32, 63].

Ways of constructing a waypoint map include e.g., visibility graphs [46], space partitioning structures such as octtrees or Delaunay triangulation [55], medial axis of the free space in the environment construction methods [88] or they may be entirely user defined.

Navigation Meshes

The idea behind navigation meshes is basically the same as the idea behind corridor maps – the navigation meshes also represent an extension of waypoint maps but instead of creating a centerline of the environment and assigning each of its points a radius, the environment is divided into a mesh (triangular, quadrilateral, or otherwise) [55, 70].

Cell and Portal Graph

Cell and portal graphs (*CPG*) introduced in [86] is a graph composed of two basic elements – the cells which represent the disjoint areas and the portals which provide the connectivity information among the cells.

11.2 Corridor Search

City Environment & Pedestrians

The complete environment for the corridor searching is composed of two parts – geometry of the city and the areas of preferences. The geometry information is used to create the navigation structures which are used for the corridor searching and the areas of preference provide the weighting information for the path-planning algorithms according to the pedestrian preferences. For the sake of simplicity, the preferences of a pedestrian to walk through each of the areas in the environment are expressed as real numbers in interval $[-1;1]$ where -1 means that the character is absolutely unwilling to walk through the area and 1 means that the character will prefer to spend as much time within the area as possible on the way towards the destination.

For our experiments, we used two different examples of urban geometry. The first of them is an artificial city structure, generated by an urban simulation and was provided to us by Prof. Bedřich Beneš from Purdue University in West Lafayette, IN. The other virtual environment is the center of Pilsen constructed from data downloaded from the Open Street Map Project [42] (*OSM*).The areas of preference were generated manually ad-hoc.

Corridor Searching Algorithms

Corridor search algorithms may be generally divided into two groups: the direct computation approach contains algorithms that compute the corridors during the runtime, when they are needed by the pedestrians and the preprocessing approach contains such algorithms that compute all the corridors that may be needed before the simulation is started. The most commonly used is the $A*$ algorithm – a representant of the direct approach. A typical example of the preprocessing approach algorithm is the *Floyd-Warshall* algorithm (*FW*) [28].

Test Cases

For our tests we used one of the two area maps displayed in Fig. 11.2(a) and Fig. 11.2(b) together with a city geometry shown in Fig. 11.2(c) which was obtained by the *EcoSim* software developed by us for this purpose. The first distribution of areas of preference (AoP) displays a center-based setting which may represent e.g., a city center surrounded by different types of neighborhood. The center is marked as the area 0, the surrounding neighborhoods are marked 1 through 4. The second distribution is a border-based example – the environment is divided into two parts by the center border area and the pedestrians have to cross the border in order to get from the top side of the environment to the bottom (and vice-versa). The dividing border area is marked as 4 in the figure, the environments are marked 0 through 3.

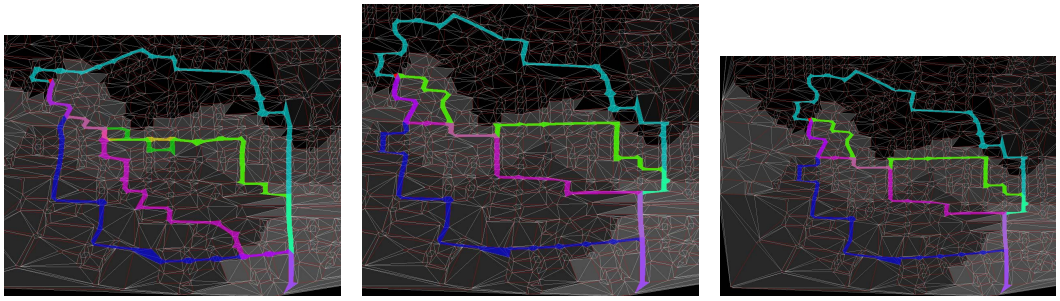
The pedestrians in the tests were distributed to five groups each with a different set of preferences for the mentioned areas. The preferences towards certain areas vary greatly among the groups of pedestrians. The two different environments and areas of preferences settings were used for testing the similarities and differences of two different corridor searching algorithm ($A*$ and Floyd-Warshall) and three different navigation data structures (navigation mesh, *CPG* and corridor map).

(a) AoP: Center-based distribution. (b) AoP: Border-based distribution. (c) City geometry.

Figure 11.2: City geometry and areas of preference.

11.3 Results

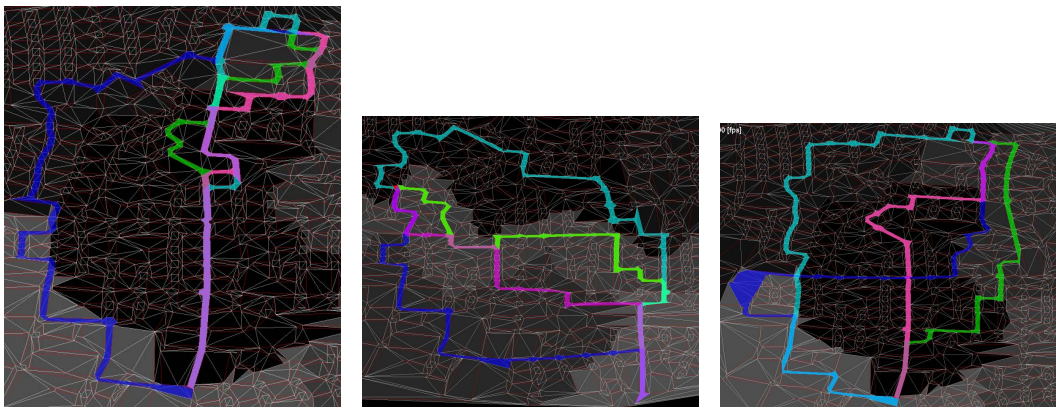
The result shown in Fig. 11.3 is composed of the border-shaped distribution of areas of interest as shown in Fig. 11.2(a). Three different corridor searches are then performed – $A*$ with a navigation mesh, *FW* with *CPG* and *FW* with a corridor map. We can see that the results provided by the *FW* algorithm are almost independent of the data structure used for the corridor search and they are not very different from the $A*$ algorithm.



(a) $A*$ with navigation mesh. (b) *FW* with *CPG*. (c) *FW* with corridor map.

Figure 11.3: Three different solutions of the border test case.

In Fig. 11.4 we can see the comparison of three different approaches for solving a test case composed of the center-based distribution of areas of preferences as shown in Fig. 11.2(a). We can see that the two approaches that use Floyd-Warshall algorithm provide us with exactly the same results while the $A*$ algorithm gives a slightly different output. Perhaps the most significant difference between the two shown results is in the behavior of the pedestrian 2 (blue color): when using the $A*$ algorithm, the pedestrians completely avoid the central area of the map which she dislikes and chooses a detour through areas 3 (top left) and 2 (bottom left) which she prefers; on the other hand, when the *FW* algorithm is used, the blue pedestrian chooses to cross the center of the map using the shortest path possible therefore maximizing the length of his route that crosses the areas 1 (top right) and 2 (bottom left). Some difference may be also observed in the behavior of the other pedestrians – pedestrian 1 (green color) chooses noticeably longer path through the center area, when the $A*$ algorithm is used, even though she prefers the top right area more than the center area. We ascribe this behavior to the fact that the $A*$ algorithm is a heuristic and the length of the path overweights the unwilingness to walk through the central area.



(a) $A*$ with navigation mesh. (b) *FW* with *CPG*. (c) *FW* with corridor map.

Figure 11.4: Three different solutions of the city center test case.

The results of our test show that the Floyd-Warshall algorithm is basically independent of the type of used data structure, the resulting corridors are very similar. This is probably because of the fact that the algorithm is exact and not a heuristic such as $A*$ that sometimes provides us with significantly different results. In general, the consequence of this difference between the tested algorithms may be expressed as the willingness to enter the area which are not preferred by the pedestrians – being a heuristic method, the $A*$ algorithm tends to find corridors that avoid the unliked areas more rigorously, especially when the corridor has to avoid (or cross) quite a large area that is not preferred by the pedestrian. In this case, the *FW* algorithm tends to find the narrowest place of such an area and cross in this place. The $A*$ algorithm is more likely to avoid the area entirely if that is possible. We think that the behavior of the $A*$ algorithm reflects reality better in this case – if we consider an example of a lone pedestrian having to get to the other side of an unlit park at night, it is more likely for them to walk around the area rather than crossing it.

11.4 Conclusion

We have shown that the algorithms commonly used for corridor search may be easily adjusted by the principle of the areas of preference in such a way that they take into account the wilingness of the pedestrians to visit or avoid certain areas within on their route towards their destination. The discussed algorithms and data structures often provide very similar corridors as results with only minor differences most of the time. However, we can say that the $A*$ algorithm generally provides us with more natural corridors – e.g., the pedestrian walks around an area that it would likely avoid when its destination is on the other side of that area without even stepping to that area, on the other hand the *FW* algorithm usually tends to cross this area (choosing the shortest possible crossing path).

Chapter 12

Conclusion

This thesis presents a compact overview of kinetic data structures, mainly the kinetic Delaunay triangulation and locally minimal triangulation with special focus on the analysis of the events in these data structures. In the first part, several possible construction algorithms for Delaunay triangulation were discussed with respect to the usability for further kinetization as well as certain auxiliary algorithms such as sublinear point location using the walk-based approach.

Further, the process of the ordinary static data structures kinetization, together with the algorithm for general management of the resulting kinetic data structures were described. Evaluation of the kinetic data structures requires us to asses their general properties, which were also outlined using the concept of certificate functions and certificate failures both in general and with special respect paid to the aforementioned kinetic triangulations. Also the most important problem in the field of kinetic data structures is discussed in detail – the computation of and the statistical analysis using advanced combinatorial methods to estimate the bounds on the number of processed kinetic events.

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

The second part of this thesis focuses on applying the previously introduced analytical methods on kinetic Delaunay triangulation and kinetic locally minimal triangulation and their mutual comparison. The computation of the kinetic events is also analyzed, general categorization of the available methods is shown, together with the most commonly used methods for polynomial equation solving as the certificate functions are most commonly polynomial functions. The concept of redundant and obsolete events was also introduced here as the research shows that large amount of the computed events is never executed. Even though the kinetic locally minimal triangulation is explored theoretically, its practical evaluation represents an open problem and should be focused on in the following research.

The focus of the third and last part of this thesis are the contributions, first of which is a modified algorithm for planar kinetic Delaunay triangulation management using the previously introduced methods for event redundancy reduction. By implementing these advanced methods,

the number of never-executed events was reduced to approximately 30%. This implementation was used in the other applications – as a geometic means of image representation in a video compression application and in the early warning system for the air traffic. The last chapter of this part described a comparison of different algorithms for large-scale pedestrian navigation which we intended to use together with kinetic Delaunay triangulation to develop a crowd simulation method.

Appendix A

Activities

A.1 Publications in Impacted Journals

[91] Tomáš Vomáčka, Ivana Kolingerová, and Martin Maňák. Kinetic locally minimal triangulation: theoretical evaluation and combinatorial analysis. *The Visual Computer*, May 2019

A.2 Publications on Web of Science and Scopus Conferences

[54] Ivana Kolingerová, Tomáš Vomáčka, Martin Maňák, and Andrej Ferko. Neighbourhood graphs and locally minimal triangulations. In *Transactions on Computational Science XXXIII*, pages 115–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018

[53] Ivana Kolingerová, Andrej Ferko, Tomáš Vomáčka, and Martin Maňák. Nearest neighbour graph and locally minimal triangulation. In *Computational Science and Its Applications – ICCSA 2017: 17th International Conference, Trieste, Italy, July 3-6, 2017, Proceedings, Part II*, pages 455–464. Springer International Publishing, Cham, 2017

[90] Tomáš Vomáčka and Ivana Kolingerová. Early warning system for air traffic control using kinetic delaunay triangulation. In Leonard Bolc, Ryszard Tadeusiewicz, Leszek J. Chmielewski, and Konrad Wojciechowski, editors, *Computer Vision and Graphics*, pages 350–356, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg

[96] Tomáš Vomáčka and Petr Puncman. A novel video compression scheme based on kinetic delaunay triangulation. In *Algoritmy 2009 : 18th Conference on Scientific Computing*, pages 372–381, Bratislava, 2009. Slovak University of Technology

A.3 Other Publications

[95] Tomáš Vomáčka and Ivana Kolingerová. On root classification in kinetic data structures. In *ADVCOMP 2011 The Fifth International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 32–35, 2011

[82] Roman Soukal, Martina Málková, Tomáš Vomáčka, and Ivana Kolingerová. Hybrid walking point location algorithm. In *ADVCOMP 2011 The Fifth International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 7–11, 2011

[94] Tomáš Vomáčka and Ivana Kolingerová. Computation of topologic events in kinetic delaunay triangulation using sturm sequences of polynomials. In *SIGRAD 2008*, pages 57–64, Linköping, 2008. University Electronic Press

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

A.4 Unpublished Manuscripts

[97] Tomáš Vomáčka, Jakub Szkandera, and Ivana Kolingerová. Comparison of the corridor selection methods for virtual pedestrian navigation. Available online: `http://bit.ly/CorridorComparison`, 2013. Unpublished manuscript

A.5 Related Talks

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

A.6 Participations in Scientific Projects

- GA17-07690S: *Methods of Identification and Visualization of Tunnels for Flexible Ligands in Dynamic Proteins*, Czech Science Foundation, 2017–2019.
- SGS-2016-013: *Advanced Graphical and Computing Systems*, University of West Bohemia, 2016–2018.
- SGS-2010-028: *Advanced Computer and Information Systems*, University of West Bohemia, 2013–2015.
- LH11006: *INGEM – Interactive Geometric Models for Simulation of Natural Phenomena and Crowds*, The Ministry of Education, Youth and Sports of the Czech Republic, 2011–2013.
- 201/09/0097: *Triangulated Models for Haptic and Virtual Reality*, Czech Science Foundation, 2009–2011.
- KJB101470701: *Alternative Representation of Image Information Using Triangulations, junior research project*, Czech Science Foundation, 2007–2009.

- LC 06008: *CPG - Center of Computer Graphics - National Network of Fundamental Research Centers*, The Ministry of Education, Youth and Sports of the Czech Republic, 2006–2011.

Bibliography

[1] Coding of audio, video, multimedia and hypermedia information. Standard ISO/IEC 14496-2:2001, International Organization for Standardization, Geneva, CH, December 2001.

[2] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, ninth dover printing, tenth gpo printing edition, 1964.

[3] P. K. Agarwal, J. Basch, M. de Berg, L. J. Guibas, and J. Hershberger. Lower bounds for kinetic planar subdivisions. *Discrete & Computational Geometry*, 24(4):721–733, Jan 2000.

[4] Air Navigation Services of the Czech Republic, Aeronautical Information Service. Various documents. Available online - http://lis.rlp.cz.

[5] 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.

[6] Mikhail Atallah. Some dynamic computational geometry problems. *Computers & Mathematics with Applications*, 11:1171–1181, 12 1985.

[7] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[8] Julien Basch. *Kinetic Data Structures*. PhD thesis, Stanford University, 1999.

[9] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.

[10] Julien Basch, Leonidas J. Guibas, Craig D. Silverstein, and Li Zhang. A practical evaluation of kinetic data structures. In *In Proc. 13th Annu. ACM Sympos. Comput. Geom*, pages 388–390. ACM Press, 1997.

[11] Leila Hashemi Beni. *Development of a 3D Kinetic Data Structure Adapted for a 3D Spatial Dynamic Field Simulation*. PhD thesis, Université Laval, Québec, 2009.

[12] Jon L. Bentley, Kenneth L. Clarkson, and David B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. *Algorithmica*, 9(2):168–183, Feb 1993.

[13] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[14] V. Carette, M.A. Mostafavi, and R. Devillers. Towards marine geographic information systems: Multidimensional representation of fish aggregations and their spatiotemporal evolutions. pages 1 –10, sept. 2008.

[15] Dan Chen, Lizhe Wang, Xiaomin Wu, Jingying Chen, Samee U. Khan, Joanna Kołodziej, Mingwei Tian, Fang Huang, and Wangyang Liu. Hybrid modelling and simulation of huge crowd over a hierarchical grid architecture. *Future Generation Computer Systems*, (0):–, 2012.

[16] 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.

[17] CS SOFT, ATM Systems & Software Development. `https://www.cs-soft.cz/`, 2019.

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

[19] Boris N. Delaunay. Sur la sphère vide. *Bulletin of Academy of Sciences of the USSR*, (6):793–800, 1934.

[20] Olivier Devillers. Improved incremental randomized delaunay triangulation. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 106–115, New York, NY, USA, 1998. ACM.

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

[22] Olivier Devillers and Olivier Devillers. Improved incremental randomized delaunay triangulation, 1997.

[23] Olivier Devillers, Sylvain Pion, and Monique Teillaud. Walking in a triangulation. In *SCG '01: Proceedings of the seventeenth annual symposium on Computational geometry*, pages 106–114, New York, NY, USA, 2001. ACM.

[24] Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. In *SCG '85: Proceedings of the first annual symposium on Computational geometry*, pages 251–262, New York, NY, USA, 1985. ACM.

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

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

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

[28] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–, June 1962.

[29] Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[30] Borivoje Furht and Borko Furht. *Motion Estimation Algorithms for Video Compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[31] 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.

[32] R. Geraerts and M. H. Overmars. The corridor map method: a general framework for real-time high-quality path planning: Research articles. *Comput. Animat. Virtual Worlds*, 18(2):107–119, 2007.

[33] 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.

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

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

[36] Leonidas Guibas and Daniel Russel. An empirical comparison of techniques for updating Delaunay triangulations. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 170–179, New York, NY, USA, 2004. ACM.

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

[38] Leonidas J. Guibas and Menelaos I. Karavelas. Interval methods for kinetic simulations. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 255–264, New York, NY, USA, 1999. ACM.

[39] Leonidas J. Guibas, Joseph S. B. Mitchell, and Thomas Roos. Voronoi diagrams of moving points in the plane. In Gunther Schmidt and Rudolf Berghammer, editors, *Graph-Theoretic Concepts in Computer Science*, pages 113–125, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[40] Leonidas J. Guibas, Feng Xie, and Li Zhang. Kinetic collision detection: Algorithms and experiments. In *ICRA*, pages 2903–2910, 2001.

[41] Stephen J. Guy, Jatin Chhugani, Sean Curtis, Pradeep Dubey, Ming Lin, and Dinesh Manocha. Pledestrians: a least-effort approach to crowd simulation. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 119–128, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[42] Mordechai (Muki) Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, October 2008.

[43] Laure Heigeas, Annie Luciani, Joëlle Thollot, and Nicolas Castagné. A physically-based particle model of emergent crowd behaviors. In *Graphicon*, 2003.

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

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

[46] H. Huang and S. Chung. Dynamic visibility graph for path planning. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2813–2818 vol.3, Sept.-2 Oct. 2004.

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

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

[49] Josef Kohout. *Delaunay triangulation in parallel and distributed environment*. PhD thesis, PhD thesis, University of West Bohemia, Pilsen, Czech Republic, 2005.

[50] Josef Kohout. Alternative representation of image information. Technical Report DCSE/TR-2009-11, University of West Bohemia, Pilsen, Czech Republic, 2009.

[51] 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.

[52] Ivana Kolingerová. Selected algorithmic metods lectures. Online resources for Selected Algorithmic Metods. http://afrodita.zcu.cz/ kolinger/vyukaZCU.html, 2019.

[53] Ivana Kolingerová, Andrej Ferko, Tomáš Vomáčka, and Martin Maňák. Nearest neighbour graph and locally minimal triangulation. In *Computational Science and Its Applications – ICCSA 2017: 17th International Conference, Trieste, Italy, July 3-6, 2017, Proceedings, Part II*, pages 455–464. Springer International Publishing, Cham, 2017.

[54] Ivana Kolingerová, Tomáš Vomáčka, Martin Maňák, and Andrej Ferko. Neighbourhood graphs and locally minimal triangulations. In *Transactions on Computational Science XXXIII*, pages 115–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018.

[55] Fabrice Lamarche and Stéphane Donikian. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, 23:509–518, 2004.

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

[57] GWENAELLE MARQUANT. *Representation par maillage adaptatif deformable pour la manipulation et la communication d'objets video*. PhD thesis, 2000. Thèse de doctorat dirigée par Labit, Claude Traitement du signal et télécommunications Rennes 1 2000.

[58] P. McMullen. The maximum numbers of faces of a convex polytope. *Mathematika*, 17(2):179–184, 1970.

[59] 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.

[60] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge; NY, 1995.

[61] Ernst P. Mücke, Isaac Saias, and Binhai Zhu. Fast randomized point location without preprocessing in two- and three-dimensional delaunay triangulations. In *SCG '96: Proceedings of the twelfth annual symposium on Computational geometry*, pages 274–283, New York, NY, USA, 1996. ACM.

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

[63] D. Nieuwenhuisen, A. Kamphuis, and M. H. Overmars. High quality navigation in computer games. *Sci. Comput. Program.*, 67(1):91–104, 2007.

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

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

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

[67] William Press, Brian Flannery, Saul Teukolsky, and William Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

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

[69] Jiří Žára, Bedřich Beneš, Jiří Sochor, and Petr Felkel. *Moderní počítačová grafika*. Computer Press, Praha, 2. edition, 2005.

[70] S. Rabin. *AI Game Programming Wisdom*. Charles River Media, inc., 2002.

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

[72] Fabrice Rouillier and Paul Zimmermann. Efficient isolation of polynomial's real roots. *J. Comput. Appl. Math.*, 162(1):33–50, 2004.

[73] Natan Rubin. On topological changes in the delaunay triangulation of moving points. *Discrete & Computational Geometry*, 49(4):710–746, Jun 2013.

[74] Daniel Russel. *Kinetic data structures in practice*. PhD thesis, Stanford, CA, USA, 2007. Adviser-Guibas, Leonidas.

[75] J.-R. Sack and J. Urrutia, editors. *Handbook of Computational Geometry*. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 2000.

[76] Raimund Seidel. The upper bound theorem for polytopes: an easy proof of its asymptotic version. *Computational Geometry*, 5(2):115 – 116, 1995.

[77] Jason Sewall, David Wilkie, and Ming C. Lin. Interactive hybrid simulation of large-scale traffic. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 135:1–135:12, New York, NY, USA, 2011. ACM.

[78] Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, 1995.

[79] Jonathan Richard Shewchuk. Sweep algorithms for constructing higher-dimensional constrained delaunay triangulations. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry*, pages 350–359, New York, NY, USA, 2000. ACM.

[80] Jiří Skála. Algorithms for manipulation with large geometric and graphic data. Technical Report DCSE/TR-2009-02, University of West Bohemia in Pilsen, 2009.

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

[82] Roman Soukal, Martina Málková, Tomáš Vomáčka, and Ivana Kolingerová. Hybrid walking point location algorithm. In *ADVCOMP 2011 The Fifth International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 7–11, 2011.

[83] R. Srikanth and A. G. Ramakrishnan. Contextual encoding in uniform and adaptive mesh-based lossless compression of mr images. *IEEE Transactions on Medical Imaging*, 24(9):1199–1206, Sep. 2005.

[84] A. Sud, E. Andersen, S. Curtis, M. Lin, and D. Manocha. Real-time path planning for virtual agents in dynamic environments. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, pages 1–9, New York, NY, USA, 2008. ACM.

[85] Mankyu Sung, Lucas Kovar, and Michael Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 291–300, New York, NY, USA, 2005. ACM.

[86] Seth J Teller. Visibility computations in densely occluded polyhedral environments. Technical report, Berkeley, CA, USA, 1992.

[87] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1160–1168, New York, NY, USA, 2006. ACM.

[88] J. van den Berg and M. Overmars. Kinodynamic motion planning on roadmaps in dynamic environments. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 4253–4258, 29 2007-Nov. 2 2007.

[89] Martin Varga. *Využití tetrahedralizace jako alternativy k objemovým datům, Diplomová práce*. Charles University, Prague, Czech Republic, 2007.

[90] Tomáš Vomáčka and Ivana Kolingerová. Early warning system for air traffic control using kinetic delaunay triangulation. In Leonard Bolc, Ryszard Tadeusiewicz, Leszek J. Chmielewski, and Konrad Wojciechowski, editors, *Computer Vision and Graphics*, pages 350–356, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[91] Tomáš Vomáčka, Ivana Kolingerová, and Martin Maňák. Kinetic locally minimal triangulation: theoretical evaluation and combinatorial analysis. *The Visual Computer*, May 2019.

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

[93] Tomáš Vomáčka. Delaunay triangulation of moving points in a plane. Master's thesis, University of West Bohemia, Univerzitní 22, Pilsen, Czech Republic, 2008.

[94] Tomáš Vomáčka and Ivana Kolingerová. Computation of topologic events in kinetic delaunay triangulation using sturm sequences of polynomials. In *SIGRAD 2008*, pages 57–64, Linköping, 2008. University Electronic Press.

[95] Tomáš Vomáčka and Ivana Kolingerová. On root classification in kinetic data structures. In *ADVCOMP 2011 The Fifth International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 32–35, 2011.

[96] Tomáš Vomáčka and Petr Puncman. A novel video compression scheme based on kinetic delaunay triangulation. In *Algoritmy 2009 : 18th Conference on Scientific Computing*, pages 372–381, Bratislava, 2009. Slovak University of Technology.

[97] Tomáš Vomáčka, Jakub Szkandera, and Ivana Kolingerová. Comparison of the corridor selection methods for virtual pedestrian navigation. Available online: `http://bit.ly/CorridorComparison`, 2013. Unpublished manuscript.

[98] 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.

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

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

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

[102] Yan Yaoping and Wu Chengke. A novel video coding scheme using delaunay triangulation. *J. Vis. Comun. Image Represent.*, 9(1):80–86, March 1998.

[103] Ya-Qin Zhang and Chang Wen Chen. *Visual Information Representation, Communication, and Image Processing*. Marcel Dekker, Inc., New York, NY, USA, 1999.

[104] Yuanfeng Zhou, Feng Sun, Wenping Wang, Jiaye Wang, and Caiming Zhang. Fast Updating of Delaunay Triangulation of Moving Points by Bi-cell Filtering. *Computer Graphics Forum*, 2010.