DYNAMIC PATH PLANNING WITH REGULAR TRIANGULATIONS

Petr Brož, Michal Zemek, Ivana Kolingerová, Jakub Szkandera
Faculty of Applied Sciences, University of West Bohemia, Pilsen, Czech Republic

Abstract. Path planning is a well known problem that has been extensively studied in many scientific disciplines. In general, it defines a task of finding a path between two given spots in an abstract environment so that the path satisfies certain criterion of optimality. Although there are many methods solving this objective, they usually assume the examined space does not change in runtime. Modern applications, however, do not have to meet these requirements, especially in case of virtual reality or computer games. Therefore, we propose a general model for real-time path planning in dynamic environment where the obstacles can nondeterministically appear, disappear, change the position, orientation or even shape. The model uses a triangulation for dynamic space subdivision among bounding spheres of the obstacles and a heuristic algorithm to repair an already found path after any change of the scene. The presented solution is the first one using regular triangulation. At the price of the suboptimal result, it provides an efficient and fast way to plan a path with the maximal clearance among the moving and changing obstacles. In comparison to raster based techniques and methods using the Delaunay triangulation (Voronoi diagram), it requires less time to preprocess and generates paths with a larger clearance.

Key words: path planning, path finding, motion planning, virtual reality, robotics, proteins, suboptimality, gaps filling.

1. Introduction

Path planning, a general problem of finding a path between two spots in a certain environment in such a way that the path satisfies one or more given objectives, is often considered to be one of the basic tasks in the computer science. However, it has been extensively studied much earlier as one of the fundamental and best known problems in the graph theory. There are therefore many sophisticated and efficient methods nowadays to solve this task in various applications, e.g., in computer networks, optimalization theory or design of data structures. In the context of the computational geometry, path planning usually involves additional procedures and characteristics such as definition, creation and maintenance of a certain representation of the environment, considering shape of the navigated entity, etc.

As there are many scientific disciplines involving the theory of path planning, various terms such as *path finding*, *route planning* or *motion planning* are sometimes used and distinguished, too. For example, the term path planning may refer to the procedure of planning an overall motion trajectory whereas the term path finding then defines the process of carrying out the path with a certain feedback for the planner. In our paper, all these terms refer to the same global problem in terms of the computational geometry.

Although the graph theory provides subtle fundamentals for many path planning algorithms,

these often suffer from serious disadvantages when they are to be used in modern applications, e.g., in the computer games or in the virtual reality (*VR*) applications. The conventional methods usually assume that the examined environment does not change in time and that its complete and detailed overview is available in advance. In the case of the modern and often indeterministic applications, it is essential to modify the existing path planning approaches or to define new ones. To provide a better solution for the mentioned type of applications, we introduce a new approach for a real-time path planning in a dynamic environment. The proposed approach is designed to suit the following requirements and characteristics:

- In terms of the dynamic environment, the new approach assumes that the obstacles can change their position, orientation or even shape in time and it is also able to register newly added or removed obstacles.
- Moreover, the behaviour of the obstacles can be either deterministic or indeterministic.
- Sometimes it is not necessary to find an optimal solution. Papers regarding the artificial intelligence actually suggest the so-called *tolerate imperfection* approach to simulate results more similar to those created by a human. On top of that, such an acceptance of non-optimal results obviously leads to more efficient solutions.

According to these specifications, the described solution is suited for a pseudooptimal navigation in an abstract environment with both static and dynamic obstacles where the obstacles may appear and disappear in time. To ensure the space subdivision among all obstacles, we use the regular triangulation with the possibility to add and remove generating points after its creation (see Section 4.1). A dual structure of the triangulation, the so-called *power diagram* (see Section 4.2), is then used to navigate among the obstacles.

The rest of the paper is organized as follows. Section 2 introduces the theoretical background of path planning and Section 3 then outlines both fundamental path planning methods and the most relevant approaches for the modern applications. Section 4 describes our proposed solution and provides a brief theoretical introduction to triangulations as they are part of the solution. In Section 5, the results of the proposed technique are presented and compared to the conventional methods. Finally, Section 6 surveys the most important characteristics of the presented algorithm and outlines possible ways of the future research.

2. Theory

As mentioned in Section 1, the term path planning in general defines a problem of finding a path between two given spots in a certain abstract environment representation [8]. In the context of the virtual reality, the term *avatar* is often used for the navigated entity.

Environment

The environment is represented in various ways (see further in this section), but all these can be generalized in the so-called *configuration space* or *c-space*, in other words, a graph $G(V,E)$ with a set of nodes V, $|V| = n$, representing all available states (in our case, positions in 2D/3D space)

and a set of edges E, $|E| = m$, defining all possible transitions between these states. The graphs are often differentiated according to various characteristics:

- Edges can be either non-directed or directed. The directed edges allow only for transitions in the given direction and non-directed edges allow for passing in both directions.
- Evaluation of the graph is ensured with an assessing function w giving each node or edge its weight.

$$w : V \to \mathbb{R} \text{ for node-weighted graphs}$$

$$w : E \to \mathbb{R} \text{ for edge-weighted graphs}$$

- Connection density – represents the number of edges in the graph. It is a very important characteristic, especially with regard to the computational complexity which can be $O(m)$ or $O(m^2)$. The graphs with a high number of edges are called *dense graphs*, whereas the graphs with a low amount of edges are usually called *sparse graphs*.
- Evaluation homogenity – distinguishes the graphs with a uniform weight distribution (the so-called *homogeneous edge costs*) and the graphs with a non-uniform weight distribution (the so-called *irregular edge costs*).

Path

The path is an acyclic sequence of the graph nodes. It is said to be optimal by satisfying one or more given objectives of optimality, e.g., the shortest path, the cheapest path or the path with the maximal clearance among all surrounding obstacles. The objectives usually result from the particular graph evaluation. For instance, the edges can be evaluated according to the time needed to traverse between their nodes or the nodes can be evaluated according to their distance from the closest obstacle.

In the area of path planning, the graphs usually have weighted and directed edges that do not form any cyclic closed path. These are called *directed acyclic graphs*.

Environment representation

The abstract environment representation for the path planning algorithms is divided into the following classes:

Graph representation – describes the given environment with a set of states and edges defining the possible transitions between the states (see Fig. 1(a)). In this case, the states represent discrete points in an n-dimensional space, e.g., the position coordinates in the 3D space.

Grid/raster representation – represents the environment with a final set of values in a matrix. Its dimension corresponds to the dimension of the represented environment. In this case, the path planning algorithms usually traverse the cells of the matrix (see Fig. 1(b)).

However, neither the virtual reality applications nor the computer games provide an environment representation immediately prepared for the path planning process. In general, the only information provided is a list of obstacles with their shapes. It is then necessary to construct the final environment representation in a certain way.

Graph representation

In the case of graph representation, the so-called *visibility graphs* (*VG*) are often used. The nodes of such a graph correspond to the vertices of all obstacles in the environment[1] and the edges connect the nodes that can "see" each other. In other words, the edges connect every pair of nodes where the line connecting these nodes does not intersect any edge (in 2D space) or face (in 3D space) of any obstacle, see Fig. 1(a). Apart from the visibility graph, other techniques and structures are used such as the *quadtrees*, the *kD-trees*, the *BSP trees* or various triangulations as we will see in Section 4.1.

Raster representation

The raster representation provides several ways to interpret the values in the matrix. The simplest way is the use of logical values to define whether there is any obstacle presented in the spot of the corresponding cell. In a more frequently used technique, the so-called *image distance transform*, the values in the matrix are specified according to the distance of a particular cell from the nearest obstacle. An information hereby received, the so-called *potential field*, then faciliates to find a path with the maximal clearance among the surrounding obstacles.

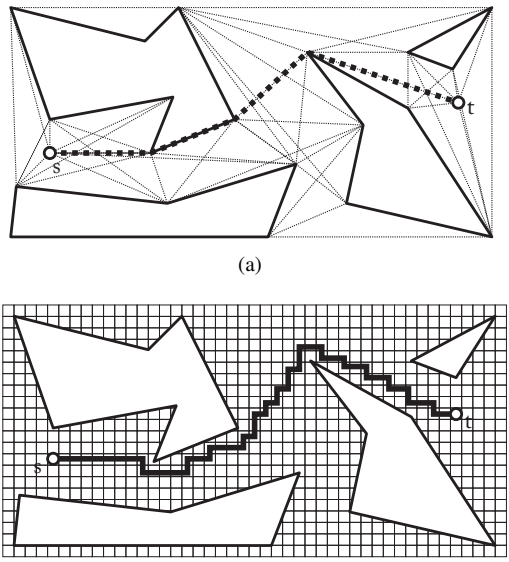


Fig. 1. (a) An example of the visibility graph. (b) An example of the grid path planning.

[1]The nodes of the visibility graph also contain the end-points of the sought path as shown in Fig. 1(a).

3. State of the art

3.1. Classification of the algorithms

As the basic path planning problem is well known since the beginning of the graph theory, there are many algorithms and techniques that need to be categorized in some way. The following list presents the most important characteristics and properties for this classification:

Extent of the desired results – determines whether a particular algorithm is supposed to return the very single path between the two given nodes (*single-pair* algorithms), to find all paths from the given source node (*single-source* algorithms) or to provide all possible paths among the graph nodes (*all-pairs shortest path* algorithms).

Environment characteristics – form the second most important factor in the classification of path planning algorithms. These characteristics are, e.g., the knowledge of the environment (known, partially-known or unknown environment), the already mentioned environment representation, its preliminary depiction etc. In case of using the c-space representation, other characteristics arise such as the possibility of a negative edge evaluation.

Avatar specification – is also a notable factor as some path planning approaches assume the navigated avatar to be represented only with a point, whereas other techniques consider its size, shape or amount of avatars at the same time.

Finding only a part of the path – is another option to be considered while planning a path in an environment that is very likely to change in time.

Suboptimal path planning – represents an approach for the applications prioritizing speed over quality.

3.2. Conventional methods

Single source algorithms

The single source (SS) algorithms are used to find optimal paths from the given source node to all other nodes of the graph. Such a solution is often obtained with the so-called *minimum spanning tree* (MST). The basic approach to construct the MST is called a *relaxation* [15].

Although the computational complexity of the MST construction is $O(2^m)$, other algorithms, obviously more efficient ones, are based just on this technique. E. W. Dijkstra formalized the problem of finding an optimal path in graph in 1956 and presented its solution for the graphs with positive edge evaluation. The solution was then generalized in 1986 into the so-called *path planning algorithm prototype*. The prototype uses a list of open nodes, that means, a list of the nodes intended for the processing. In the beginning of the prototype algorithm, the list is empty and the only node inserted is the start node of the path. As long as there are any nodes in the list, the algorithm chooses one of them (to be more precise, the node is removed from the list) and checks whether this node provides a shorter path to any of its adjacent nodes. The approach is outlined in Algorithm 3.1 where the implementation of methods `Insert` and `Select` is subject to specific modifications in the particular techniques.

Algorithm 3.1 Path planning algorithm prototype

Input: *graph* contains a final set of nodes and edges {Each node contains a reference to the preceding node in the searched path (*node.parent*) and a value defining its minimum distance from the starting point of the path to the current node (*node.distance*); each edge contains reference to its end-points (*edge.start* and *edge.end*)}
Input: *root* is one of the nodes in *graph*
Input: *list* is an empty list of nodes

```
list.Insert(root)
while list is not empty do
  node ← list.Select() {The selected node is removed from the list}
  for all edge in graph.edges do
    dist ← edge.start.distance + distance_between(edge.start, edge.end)
    if edge.end.distance > dist then
      edge.end.parent ← edge.start
      edge.end.distance ← dist
      list.Insert(edge.end)
    end if
  end for
end while
```

Dijkstra's algorithm does not need the `Insert` method and uses the so-called *greedy* approach to select the node to be processed next – the algorithm always uses the node with the minimal distance from the source node. If a sequential search is used to find the closest node, the overall computational complexity is $O(n^2)$. Therefore, various data structures have been presented to speed up the technique, e.g., the *d-Heaps* [5]. For the graphs where the negative evaluation is also allowed, the Bellman-Ford's algorithm is used [9] with the complexity $O(mn)$. It however tends to the complexity $O(m^3)$ for very dense graphs according to the definition from Section 2.

All-pairs shortest path algorithms

The all-pairs shortest path (*APSP*) algorithms find the optimal paths between every pair of nodes in the graph. As the construction of a MST for every single node would be very inefficient, the so-called *all-pairs shortest distance* algorithms are rather used to provide only the lengths of the optimal paths between each pair of nodes. It is possible to apply the Dijkstra's algorithm or the Bellman-Ford's algorithm, respectively, for each node of the graph, however, with the overall computational complexity $O(n^3)$, $O(n^4)$, respectively. A better solution for this task, Floyd-Warshall's algorithm [9], works with the so-called *distance matrix* and *matrix of precursors*.

It is known that the Floyd-Warshall's algorithm works with the complexity $O(n^3)$. It is also possible to use this approach for the graphs with negative evaluation of the edges but without any cycles. However, improving the computational complexity turned out to be a significant

problem. An algorithm combining the Dijkstra's and Bellman-Ford's algorithm has been presented in 1977. The improvement works in $O(n^2 \log n + mn)$ under the assumption of a sparse and oriented graph without any cycles. Without this assumption, the only way to break the cubic complexity is an approximation of the desired path, often denoted as the *t-optimal path*. Such a path can be at most n-times worse than the optimal solution. For instance, Don, Halperin and Zwick presented an efficient algorithm for finding a 3-optimal path [12].

Single pair shortest path algorithms

The single pair shortest path (*SPSP*) algorithms form the basic class in the whole problematics of path planning. In this case, only a single path between the two given nodes is required. The best known techniques in this area [9] are the *depth first search* (*DFS*) and the *breadth first search* (*BFS*). From the theoretical point of view, the single source algorithms and the single pair algorithms have the same computational complexity. However, the single pair algorithms are usually more efficient as the single solution is found earlier.

A* algorithm [15], the most significant representative in this class of algorithms, uses certain heuristics to underestimate the nodes selected for the next processing. In this case, the heuristic function is defined as $f(u) = g(u) + h(u)$, where $g(u)$ is the minimum distance between the start node and the currently selected node $u \in V$, $h(u)$ is the heuristic defined by the Euclidean distance from the end node to the currently selected node $u \in V$. A node that is closer to the destination has a bigger chance of being selected for the next processing. The A* algorithm is also generalized into the so-called *best-first search*.

3.3. Advanced methods

In this section, we briefly describe the most important representatives from the wide range of advanced path planning applications available nowadays.

D* algorithm

The D* algorithm [6] modifies the A* algorithm and is suited for graphs whose evaluation may change in time of its traversal. It is also suited for unknown, partially known or varying environments. The D* algorithm uses the same list of open nodes as the single source techniques and propagates every change of evaluation throughout this list. Each node keeps an information about the smallest estimated distance to the destination node called *key function*, which is also used to sort the nodes in the list. Any evaluation change is then propagated to all appropriate key functions and causes reordering of the nodes in the list. The algorithm provides an optimal solution by planning a new path after each change in the environment. Although this approach corresponds to the brute-force solution, it is more effective.

Path planning system for strategic games

A good example of a path planning approach for navigating several avatars at the same time has been presented in [14]. It distinguishes static obstacles, avatars waiting for the orders and avatars that are currently in motion. In the preprocessing time, the visibility graph (see Fig. 1(a))

is created for the static obstacles. The waiting avatars are gathered into clusters and the initial path found in the graph is then adjusted to avoid the collision with any of these clusters. In the time of executing the movement, possible collisions between the moving entities are avoided by temporarily stopping one of these entities until its path is free again.

Space discretization for efficient human navigation

The approach proposed in [11] uses the rasterization technique described in Section 2. The input scene is discretized into a certain matrix consisting of cells of the same size, the so-called *uniform cells*. Inside these cells, the optimal path is then found using the mentioned A* algorithm. As the desired path should serve for human-like avatars, the basic criterion for the resulting path is its accessibility for the humans. In other words, the path cannot lead through the air. An example of such a path is shown in Fig. 2(a).

Probabilistic roadmaps approach

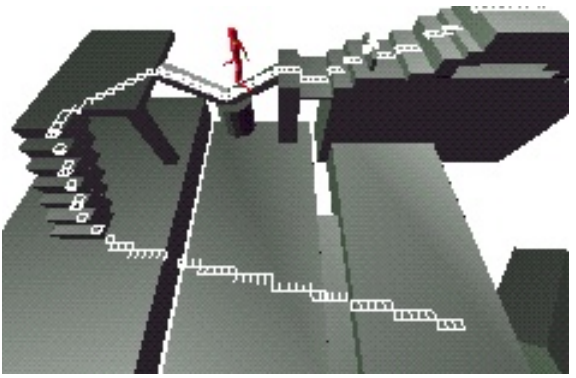
The probabilistic roadmap [16] is one of the frequently used techniques nowadays. The so-called *probabilistic roadmap planner* randomly samples the input scene for collision-free positions and adds these positions as nodes to a specific roadmap graph structure (see Fig. 2(b)). In the following step, the planner chooses various pairs of *promising nodes* and attempts to connect them with a local planner until the c-space is fully covered. The exact implementation details (e.g., the space sampling, the selection of promising nodes) are then subject to particular variants of this path planning approach.

Voronoi-based route graphs

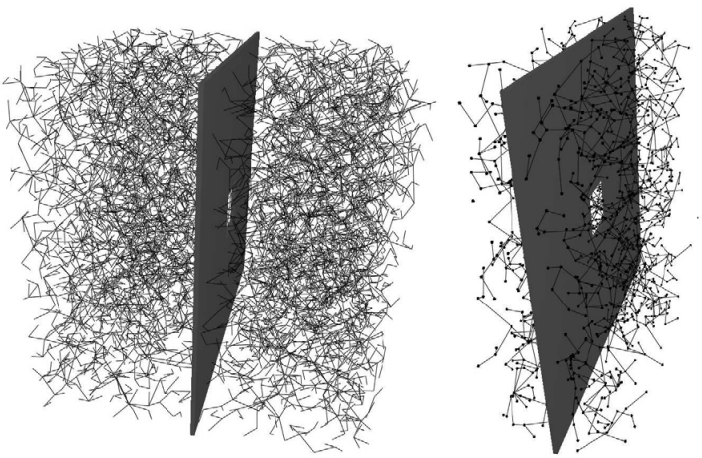
The hierarchical Voronoi-based route graph [20] defines a spatial representation of the environment for the purposes of robot navigation. For any input scene consisting of polygonal obstacles, a *generalized Voronoi diagram* (*GVD*) is constructed as shown in Fig. 3 and its edges are then used for the navigation along the path with the maximal clearance among all the obstacles. Such a construction usually consists of creating a particular *Voronoi diagram* (*VD*) for the vertices of every single obstacle in the scene and consequently removing all the unnecessary edges.

Grid and graph representation

As the last example of modern solutions in the area of path planning, we present our previous research [23] which is compared to the newly proposed solution in Section 5. The approach developed in the previous research combines both raster and graph representations of the environment. While the raster defines a certain danger (in this case, the danger is represented by the proximity to the nearest obstacle and therefore by the probability of a collision) of the scene in particular positions (see Figs 4(a), 4(b), the graph as an adaptive spatial structure refines or coarses itself according to this danger in the corresponding areas (see Fig. 4(c)). The adaptive graph structure then represents the c-space and allows the approximate navigation along the least dangerous path.



(a)



(b)

Fig. 2. (a) A discretized path planning for human-like avatars [11]. (b) An example of a space sampling using the probabilistic roadmaps [16].

4. Proposed solution

In the case of dynamic applications of virtual reality or computer games, not even the modern path planning methods often provide a sufficient solution, mainly because of the following reasons:

- in our focus, we assume the navigation to be executed in E^3 space. This assumption may

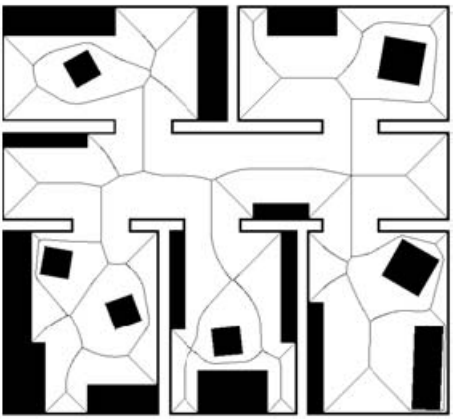
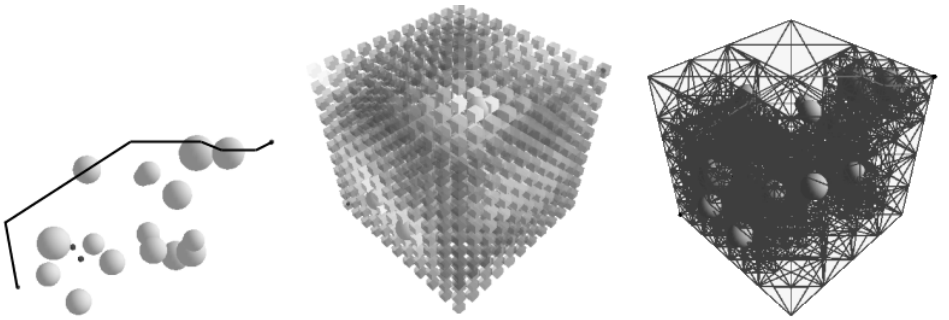


Fig. 3. Generalized Voronoi diagram for an example scene [20]



(a) (b) (c)

Fig. 4. (a) An example scene. (b) Its raster representation. (c) An adaptive mesh refined according to the raster representation in Fig. 4(b).

cause serious complications in the techniques based on a discrete representation, especially in the terms of memory consumption;

- the modern algorithms based on the graph representation usually construct the c-space according to the exact shapes of the obstacles. As it may significantly affect the computational complexity, we prefer to provide an efficient solution at the cost of the optimality;
- some of the methods described in Section 3.3 are not able to operate in an undeterministic environment which is obviously a very frequent case in the VR applications.

Therefore, we propose a solution that uses a triangulation to navigate among bounding

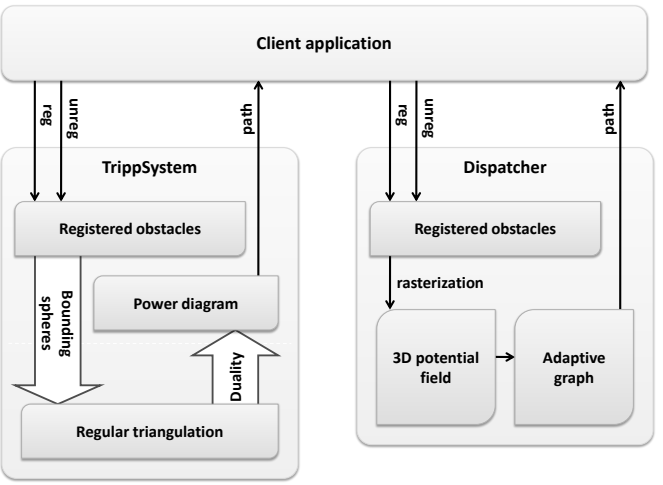


Fig. 5. A design of the proposed path planning solution.

spheres of the obstacles in the scene. It is a prime approach in using the *regular triangulation* (*RT*) and the correspondent dual structure, the *power diagram* (*PD*), instead of the Voronoi diagram. As we will show in Section 4.1, the RT makes it posible to define certain weights of the points that generate the triangulation. In our proposal, the generating points and their weights are defined by each bounding sphere. The corresponding PD then defines the c-space and allows us to plan a path with the maximal clearance among the bounding spheres. Moreover, the implemented RT provides the possibility to insert, remove and change the generating points in the runtime, thus allowing us to react to any change in the scene. The overall design of this approach is outlined in Fig. 5. The block labeled *TrippSystem* represents the new approach based on the triangulations (triangulation based path planning system), whereas the second block labeled *Dispatcher* represents our previous solution combining raster and graph representations of the environment, see the last paragraph in Section 3.3.

The main process of planning a path in the c-space is ensured by the so-called *gaps filling algorithm* that will be described in Section 4.3. After any change in the scene, the gaps filling algorithm repairs the previously found path instead of planning a new one from the scratch.

We provide a brief theoretical background for the regular triangulations and related structures (Sections 4.1, 4.2) and describe the heuristic algorithm used to maintain a suboptimal path in the varying environment (Section 4.3).

4.1. Regular triangulation

The regular triangulation (RT) is a generalization of well-known Delaunay triangulation (DT). Each point in the regular triangulation is associated with a real number – a weight of the point. If the weights of the points are equal, then the regular triangulation and the Delaunay triangulation of this set of points are identical, otherwise the triangulations can be different. Further we give a definition of the regular triangulation (see [4]).

Triangulation – Given a set of points S in E^3, the triangulation $T(S)$ of this set of points is a set of tetrahedra such that:

- a point $p \in E^3$ is a vertex of a tetrahedron in $T(S)$ only if $p \in S$;
- the intersection of two tetrahedra of $T(S)$ is either empty or it is a shared face, a shared edge or a shared vertex;
- the union of all tetrahedra in $T(S)$ entirely fulfills the convex hull of S.

Note that in this definition of $T(S)$ we do not require each $p \in S$ to be a vertex of $T(S)$.

Weighted point – A point $p \in E^3$ with an associated weight $w_p \in \mathbf{R}$ is called a weighted point. If the weight w_p is non-negative, then p can be interpreted as a sphere centered at the point p with a radius $\sqrt{w_p}$.

Power distance – A power distance of a weighted point p from a point $x \in E^3$ (no matter whether x is weighted or unweighted) is defined as $\pi_p(x) = |px|^2 - w_p$, where $|px|$ denotes the Euclidean distance between the points p and x. The power distance $\pi_p(x)$ can be interpreted as a square of length of a tangent from the point x to a sphere centered at p and with the radius $\sqrt{w_p}$ (if x lies outside this sphere), see Fig. 6a.

Orthogonal points – Two weighted points p and q are said to be orthogonal if $|pq|^2 = w_p + w_q$, i.e. $\pi_p(q) = w_q$, see Fig. 6b.

Orthogonal center – Let a, b, c, d be non-coplanar weighted points. A weighted point z is an orthogonal center of a tetrahedron $abcd$ if z is orthogonal to the points a, b, c, d.

Global regularity – Let z be the orthogonal center of a tetrahedron $abcd$. The tetrahedron $abcd$ is globally regular with respect to a set of weighted points S if $\pi_z(p) > w_p$ for each point $p \in S - \{a, b, c, d\}$.

Regular triangulation – A triangulation $T(S)$ is a regular triangulation of S ($RT(S)$) if each tetrahedron in $T(S)$ is globally regular with respect to S.

Redundant point – A point p, $p \in S$, is called a redundant point if no globally regular tetrahedron $pabc$ exists in $RT(S)$. Redundant points are not vertices of any tetrahedron in $RT(S)$, therefore, the vertex set of $RT(S)$ is generally only a subset of S.

Basic characteristics

Similarly to the Delaunay triangulation, the regular triangulation of S is unique if the points of S are in a general position. In contrast to DT, it is possible that some point $p \in S$ is not a vertex of $RT(S)$. From the given definition of $T(S)$ it follows that each point $p \in S$ lying on the convex hull $CH(S)$ of S is a vertex of $RT(S)$ – otherwise the union of tetrahedra would not fulfill $CH(S)$.

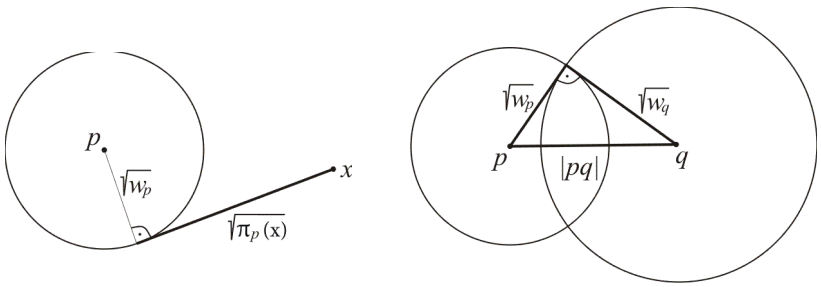


Fig. 6. (a) The power distance $\pi_p(x)$. (b) Two orthogonal weighted points.

Shewchuk [19] proved that in the worst case the number of tetrahedra of $RT(S)$ is $O(n^2)$, where $n = |S|$. This happens if the points of S lie on (or nearby) two nonintersectors. If the points of S are uniformly or nearly uniformly distributed, the expected number of tetrahedra grows with n nearly linearly.

Dynamic triangulation

As already mentioned, our intended scene is dynamic, thus obstacles can appear, disappear, discretely change their size or position. And an underlying regular triangulation must handle such events. It is not necessary to rebuild a whole triangulation after each change. All the four types of changes of an obstacle (a weighted point) in the triangulation can be implemented as an insertion, deletion or a combination of both (when an obstacle changes its size or position, it is first removed from the triangulation and then re-inserted). Therefore we use an algorithm of an incremental insertion (see e.g. [4, 1]) for the triangulation construction. Here, points are inserted into the regular triangulation one by one and the regularity of the triangulation is restored after each insertion. The time complexity of this algorithm is $O(n^2)$ in the worst case. In the average case, the time complexity is of our implementation is $O(n^{5/4})$.

For the deletion of a point from a triangulation, we employ the algorithm described in [18]. This algorithm removes all tetrahedra incident to a point to be deleted. This creates a cavity - a star-shaped polyhedron P. The cavity is then retriangulated by a successive cutting of ears of the P. The time complexity of the deletion of one point is $O(kg)$, where k is a number of vertices of P and g is a number of tetrahedra created to retriangulate P. Note that g is $O(k^2)$ in the worst case and $O(k)$ in the average case. Other applicable algorithms of a point removal are described in [17, 26, 22].

4.2. Power diagram

Power cell – Given a set of weighted points S in E^3. For each weighted point $p \in S$, its power cell is defined as $power\ cell(p) = \{x \in E^3 | \forall q \in S - \{p\} : \pi_p(x) \leq \pi_q(x)\}$. The point p is

the so-called *generator* of *power cell*(p). Observe that *power cell*(p) is a convex polyhedron and the union of all *power cells*(p), $p \in S$, covers E^3. An intersection of two or more power cells is either empty or forms:

- a planar convex polygon – a face of $PD(S)$. A face of $PD(S)$ is the intersection of two power cells;
- a line segment or a half line – an edge of $PD(S)$. An edge of $PD(S)$ is the intersection of at least three power cells;
- a point – a vertex of $PD(S)$. A vertex of $PD(S)$ is the intersection of at least four power cells and the sphere orthogonal to the generators of these power cells is centered in this vertex of $PD(S)$.

Power diagram – The power diagram of S is a collection of all power cells, their faces, edges and vertices.

Thanks to the duality, power diagrams could be seen as another representation of regular triangulations and vice versa. Also each of these structures can be derived from the other in linear time. Any computation performed on one of these structures can be also done on the other structure and with the same time complexity. For example, a path in $PD(S)$ – a sequence of vertices and edges – can be represented as a sequence of tetrahedra and faces in $RT(S)$.

From the point of view of the path planning, the power diagrams (in comparison with the ordinary Voronoi diagrams) offer one big advantage. Let the power cells of two weighted points $p,q \in S$ share a face. If we interpret the points p,q as spheres with the radii $\sqrt{w_p}, \sqrt{w_q}$, respectively, and these spheres do not intersect each other, then the shared face does not intersect any sphere from S.

Power diagram and c-space

We interpret a power diagram $PD(S)$ as a c-space $G(V,E)$ as follows:

- each edge $h \in PD(S)$ corresponds to one node $v_h \in V$;
- two nodes $v_h, v_k \in V$ are connected by an edge $e \in E$ if and only if the two corresponding edges $h,k \in PD(S)$ share a common vertex;
- a weight w of a node of $v_h \in V$ is computed as

$$w = \frac{1}{1+d},$$

where d is a euclidean distance between the edge $h \in PD(S)$ and the closest weighted point $p \in S$, i.e.,

$$d = min_{x \in h, p \in S}\{|xp| - \sqrt{w_p}\}.$$

Therefore, $w \in \langle 0,1 \rangle$ and a value of w corresponds to a probability of a collison. An exact computation of the distance d is not trivial and in the worst case can take $O(n)$ time, therefore we approximate d by a radius of a so-called bottleneck sphere of h. A bottleneck sphere of h is the biggest sphere, which can travel along h from one its endpoint to the other without intersecting

generators of power cells forming h (for details, see [27]). The computation of a bottleneck sphere is a $O(1)$ operation. The drawback is that a bottleneck sphere still can intersect some generators from S. But sizes of such intersections tend to be small. Note that if the approximate distance is positive, the exact distance must be also positive.

4.3. Gaps filling

The gaps filling algorithm [24] provides a fast way to find a suboptimal path in graphs with varying topology or evaluation. Instead of finding a new path after any change in the graph, this approach repairs the last found path along the nodes that have been removed and the nodes whose evaluation has been made worse. An example of such a procedure is presented in Fig. 7. First, a new path was found between the nodes s,t using the standard Dijkstra's algorithm and in the following iteration, after worsening of the evaluation in the nodes a,b, these nodes have been omitted and the resulting path segments have been connected again using the Dijkstra's algorithm. The procedure is outlined in Algorithm 4.1.

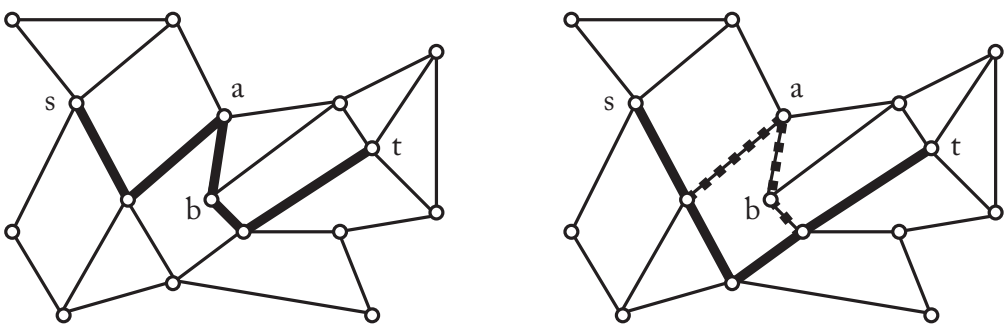


Fig. 7. An example of the gaps filling approach [24].

At the cost of the suboptimal solution, the algorithm offers a significant speed-up in terms of the number of processed nodes during the search. Tab. 1 shows the speed-up compared to the standard Dijkstra's algorithm – in a planar graph of 32x32 nodes which are randomly connected to some of the neighbouring nodes, the evaluation has been changed in a certain percentage of the nodes. The second column shows the percentage of nodes processed for re-planning the path by the gaps filling algorithm in comparison with the Dijkstra's algorithm. The last column then specifies the percentual change of the overall path quality, an average of weights of all nodes on the path where the minimum weight is the goal.

Algorithm 4.1 Gaps filling algorithm

Input: a graph $G(V,E)$, a path $P \subset V$, ending nodes $s,t \in P$
Output: new path P'

```
P' ⇐ {s}
u ⇐ NULL {Auxiliary node}
for all n ∈ P − {s} do
  if n was removed ∨ n has worse evaluation then
    if u == NULL then
      u ⇐ predecessor of the node n on the path P
    end if
  else
    if u == NULL then
      P' ⇐ P' ∪ {n}
    else
      P' ⇐ P' ∪ path(u,n) {Finding a path from u to n using Dijkstra's algorithm}
      u ⇐ NULL
    end if
  end if
end for
```

Tab. 1. Speed-up of the gaps filling algorithm.

% nodes changed	% nodes processed	% weight of path
25	13.70	126.26
50	37.75	115.76
75	85.67	108.67

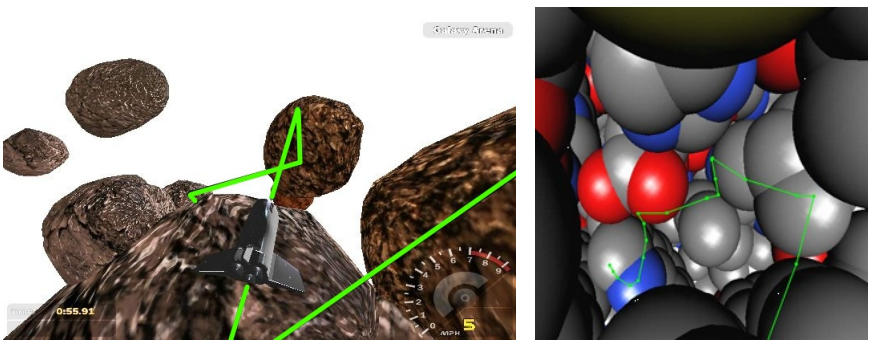
5. Experiments and results

5.1. Implementation

The proposed path planning solution was implemented as a dynamically linked library in the C# language in order to enable its easy use in modern applications while allowing us to continue with efficient development and debugging. For testing purposes, a simple game application was created, too, using the XNA Game Studio. The demo simulates a spaceship flying through a swarm of asteroids where some of these asteroids are moving, see Fig. 8(a). The ship may be controlled by the user or by an automatic navigation, in other words, by our implemented approach. In addition to the visual presentation, the game application was also used to measure

and gather various characteristics of the path planning algorithm for different types of behavior of the environment. The characteristics are presented and described in the following sections.

The proposed method was also tested in the context of real biochemistry data. To be more precise, the technique described in this paper was used to plan a path in a dynamic environment of protein molecules, see Fig. 8(b). In this case, movement of the proteins was defined in a discrete way by providing their location in different time slices. Results of this measurement are presented in Section 5.5.



(a) (b)

Fig. 8. (a) A screenshot from the testing application Galaxy Wars. (b) An example of a path in a protein molecule.

In case of the path planning algorithm itself, we have tested several options (with the correspondent notation in the following charts):

- Type of the triangulation for the space subdivision:
  - regular triangulation (denoted with *RT*);
  - Delaunay triangulation (denoted with *DT*).
- Accuracy of the so-called *in-circle test* used in the triangulation alteration:
  - exact arithmetics (denoted with *EA*);
  - non-exact arithmetics (denoted with *NEA*).

Finally, the following list summarizes the most important characteristics we have observed during the tests:

Preprocessing time – It is often possible and very efficient to preprocess the available information about the environment and we have considered this property for all the tested algorithms.

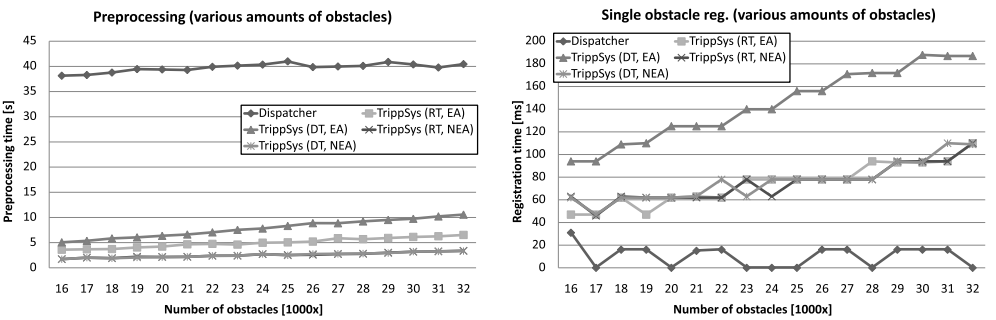
Single registration time – In our proposal, the adaptation to the changes of the scene is ensured by re-inserting the generating point of the regular triangulation (see Section 4). We have therefore measured the time needed to insert a single point into the triangulation.

Path weight – An average path weight is defined as the arithmetic mean of all weights of the path nodes. Weight of any node represents its proximity to the nearest obstacle, higher weight

means a higher danger of collision with the obstacle. The maximum danger value of 1.0 then defines a collision state with the obstacle.

5.2. Various numbers of obstacles

Dependence of the speed and quality of the provided results on the number of obstacles is obviously the most important characteristic in the context of path planning algorithms. In Figs 9(a) and 9(b), we present the preprocessing and registration times for various configurations of the proposed solution (TrippSys) together with the solution based on graph and grid representation of the environment (Dispatcher, see Section 3.3). It can be seen that the triangulation based techniques provide shorter preprocessing times than the conventional approach, even for higher numbers of obstacles. On the other hand, the time needed to insert an obstacle to the triangulation that already consists of a given number of generating points is greater than in the conventional case. In Dispatcher, every single obstacle is projected on the grid structure and consequently causes the adjustment of the adaptive graph. The time needed to insert an obstacle in such a manner therefore does not change with the increasing number of already registered obstacles.

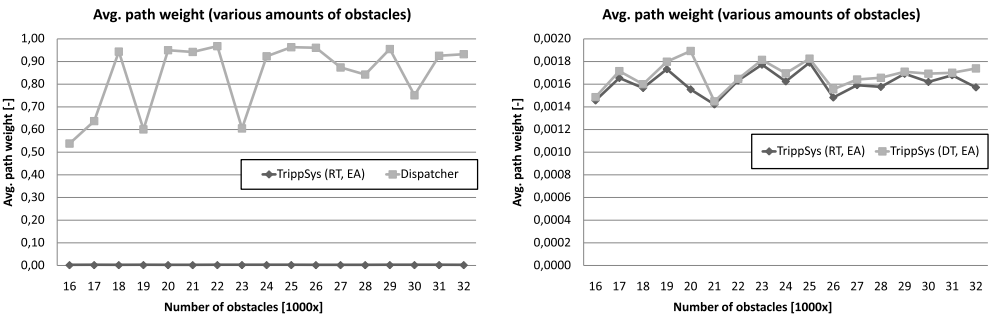


(a) (b)

Fig. 9. (a) Preprocessing times for various numbers of obstacles in the scene. (b) Single obstacle registration times for various numbers of obstacles already in the scene.

As for the quality of the found path, Figs 10(a) and 10(b) show the average path weight for the increasing number of obstacles registered in the system. Here we can see the most significant advantage of the triangulation based path planning algorithms. The conventional approach based on the combination of graph and raster representation navigates along the edges of the adaptive graph that does not provide many possibilities of movement, thus providing a less safe path. On the other hand, the edges of the power diagram are always safe in terms of the distance from the surrounding obstacles. Fig. 10(b) also shows that the Delaunay triangulation (its dual structure, Voronoi diagram, respectively) provides slightly worse results – it only

considers the position of the generating points, whereas the regular triangulation also considers their weight represented by the radius of the bounding spheres. Therefore, the path provided by the dual structure of the regular triangulation has the maximal clearance among the obstacles.

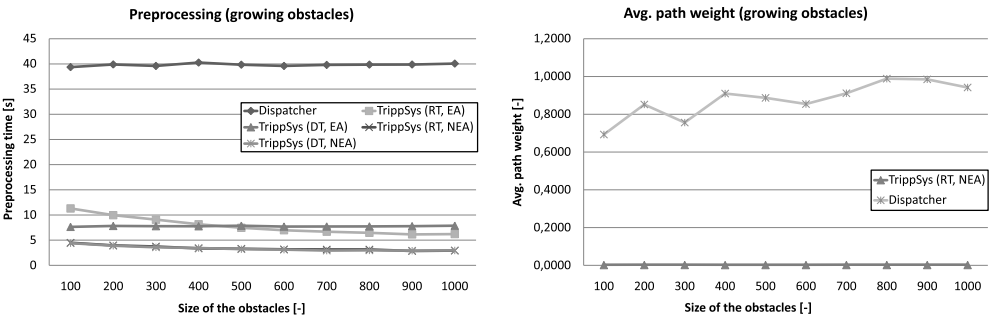


(a) (b)

Fig. 10. (a) Average path weights for various numbers of obstacles in the scene (compared to the raster based algorithm). (b) Average path weights for various numbers of obstacles in the scene (using the triangulation based algorithms).

5.3. Growing obstacles

In the following set of tests, we have observed the behavior of the chosen approaches in an environment consisting of a constant number of obstacles whose radii have been increased in several steps. In the last step, the obstacles almost fully covered the examined space. For the environment of 24000 obstacles with various radii, Fig. 11(a) shows the preprocessing times and Fig. 11(b) then outlines the average weight of the found path in each technique. It can be seen that the change of the size does not affect the characteristics in any significant manner. The triangulation based approaches again demand more time for the registration of a single obstacle but provide significantly better results in terms of the path quality. The average weight of the path found by the conventional algorithm Dispatcher higly varies due to the fact that this approach depends strongly on the coverage of the space with the obstacles. For instance, Fig. 11(b) shows a worsening of the path in the beginning (size 100 to 200) immediately followed by an improvement (size 200 to 300) – as the obstacles increase their radii, bigger part of the examined space is covered which in consequence causes a refinement of the adaptive graph structure in that location, thus providing a finer and safer path.



(a) (b)

Fig. 11. (a) Preprocessing times for growing obstacles. (b) Average path weights for growing obstacles.

5.4. Obstacles in a dilating cluster

In case of a single growing cluster consisting of a constant number of obstacles, the characteristics of the preprocessing and registering time were again very similar to the previous measurements. Fig. 12 however shows an interesting dependence of the path quality while increasing the radius of the cluster of 24000 obstacles. With higher coverage of the space with the obstacles, the conventional solution provides worse results whereas the triangulation based algorithm provides slightly better results. The explanation again lies in the behavior of the space division structures used in both approaches. If only a small part of the space is covered with a cluster of obstacles, the raster based method chooses a path around this cluster. On the other hand, the triangulation based technique has to follow the edges of the power diagram leading through the cluster. By increasing the radius of the cluster, the safe surroundings for the conventional algorithm disappear whereas the edges of the power diagram leading among the obstacles gain higher freedom.

5.5. Protein molecules

In the last step of our measurement, we have observed the path planning systems in an environment consisting of real protein data. The proposed approach proved to be an efficient solution in terms of the preprocessing time and the resulting path. Figs 13(a) and 13(b) present the particular results for various time slices of real protein data and various numbers of molecules.

6. Conclusion

The presented path planning approach is the first one using the regular triangulation for the space subdivision. In comparison to the raster based techniques and the methods using the Delaunay

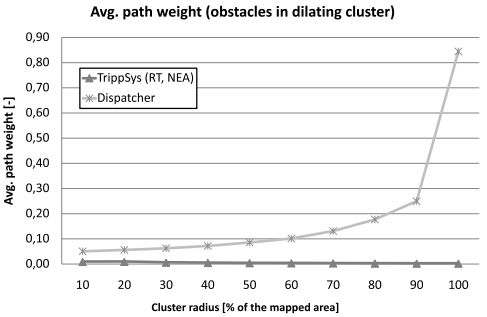
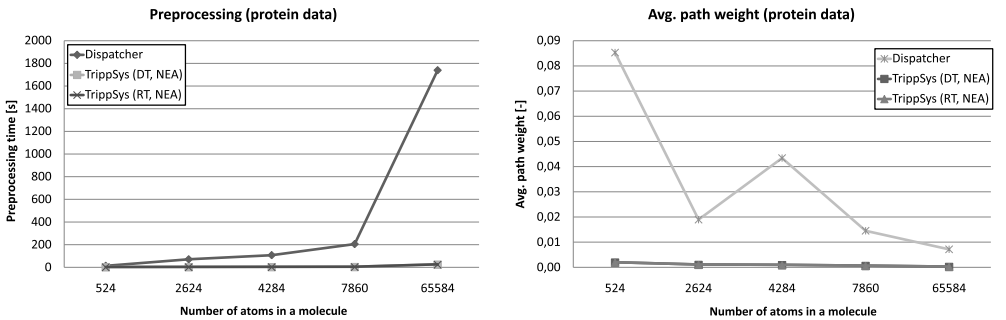


Fig. 12. Average path weights for a dilating cluster of obstacles



(a) (b)

Fig. 13. (a) Preprocessing times for the protein data. (b) Average path weights for the protein data.

triangulation (Voronoi diagram), it requires less time to preprocess and provides significantly safer path (in terms of the distance from the surrounding obstacles) at the cost of suboptimal requirements for the additional registration of a single obstacle. Unlike many path planning techniques, the proposed solution is able to adapt to any change in the scene – insertion/removal of any obstacle, position change, shape change etc. – by only recalculating the correspondent bounding sphere and reinserting the correspondent generator in the triangulation.

Moreover, the proposed gaps filling algorithm (Section 4.3) is able to substantially speed up the main process of replanning the path while providing reasonable suboptimal results. In an environment of 2^{16} randomly positioned and moving obstacles, the gaps filling approach provided a slightly suboptimal solution (1.1-optimal on the average) while taking only 20% of the time needed by the Dijkstra's algorithm.

For the continuing research, we have pointed out the most important and most interesting ways of the improvement:

- as stated in Section 4, the current solution finds a path among the bounding spheres of the obstacles. In the future research, it is our intention to consider this path as the first estimation and locally refine it according to the particular shapes of the surrounding obstacles;
- the proposed technique does not consider any shape of the navigated entity so far. One of the future aims would therefore be the retrieval of a certain tunnel instead of a path for navigating an avatar with concrete proportions.

Acknowledgments

This work was supported by the Ministry of Education, Youth and Sports of Czech Republic - project Kontakt No. LH11006, by the UWB grant SGS-2013-029 – Advanced Computer and Information Systems, and by the Ministry of Education, Youth and Sport of Czech Republic – University spec. research – 1311.

References

1981

[1] Watson D. F.: Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes, The Computer Journal, 2:167-172.

1986

[2] Fortune S.: A sweepline algorithm for Voronoi diagrams, SCG'86: Proceedings of the second annual symposium on Computational geometry, p. 313-322.

[3] Aurenhammer F.: Power Diagrams: Properties, Algorithms and Applications, SIAM Journal on Computing, 16:78-96.

1992

[4] Edelsbrunner H., Shah N. R.: Incremental Topological Flipping Works for Regular Triangulations, ACM Annual Symposium on Computational Geometry, 8:43-52.

1994

[5] Eppstein D.: Finding the k-Shortest Paths, 35th Annual Symposium on Foundations of Computer Science, p. 154-165.

[6] Stentz A.: Optimal and Efficient Path Planning for Partially-Known Environments, Proceedings IEEE International Conference on Robotics and Automation, San Diego, California, USA, p. 3310-3317.

1996

[7] Barber C. B., Dobkin D. P., Huhdanpaa H.: The quickhull algorithm for convex hulls, ACM Trans. Math. Softw., 22:469-483.

1997

[8] Goodman J. E., O'Rourke J.: Handbook of Discrete and Computational Geometry, CRC Press LLC.

[9] Skiena S. S.: The Algorithm Design Manual, Springer-Verlag, New York.

[10] Cignoni P., Montani C., Scopigno R. : DeWall: A fast divide and conquer Delaunay triangulation algorithm in E^d, Computer-Aided Design, 30:333-341.

1998

[11] Bandi S., Thalmann D.: Space Discretization for Efficient Human Navigation, Computer Graphics Forum, 17(3):195-206.

2000

[12] Dor D., Halperin S., Zwick U.: All-Pairs Almost Shortest Paths, SIAM Journal on Computing, 29(5):1740-1759.

[13] Shewchuk J. R.: Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations, SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry, p. 350-359.

2001

[14] Arikan O., Chenney S., Forsyth D. A.: Efficient Multi-Agent Path Planning, Computer Animation and Simulation, p. 151-162.

[15] Champandard A. J.: Path-Planning from Start to Finish (Internet source: `http://www.ai-depot.com/BotNavigation`).

2002

[16] Geraerts R., Overmars M. H.: A Comparative Study of Probabilistic Roadmap Planners, Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02), p. 43-57.

[17] Vigo M., Pla N., Cotrina J.: Regular Triangulations of Dynamics Sets of Points. Computer Aided Geometric Design 19:127-149.

2003

[18] Devillers O., Teillaud M.: Perturbations and Vertex Removal in a 3D Delaunay Triangulation. Proceedings 14th ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, USA, p. 313-319.

2004

[19] Shewchuk J.R.: Stabbing Delaunay tetrahedralizations, Discrete Comput. Geom., 32:343.

2005

[20] Wallgrün J. O.: Autonomous Construction of Hierarchical Voronoi-Based Route Graph Representations, Spatial Cognition IV. Reasoning, Action, and Interaction, 3343:413-433.

[21] Beyer T., Schaller G., Deutsch A., Meyer-Hermann M.: Parallel dynamic and kinetic regular triangulation in three dimensions, Computer Physics Communications, 172:86-108.

[22] Ledoux H., Gold C.M., Baciu G.: Flipping to Robustly Delete a Vertex in a Delaunay Tetrahedralization. ICCSA (1):737-747.

2007

[23] Brož P., Kolingerová I., Zitka P., Apu R. A., Gavrilova M. : Path planning in dynamic environment using an adaptive mesh, Proceedings of the SCCG, p. 172-178.

[24] Brož P. : Exact and heuristic path planning methods for a virtual environment, Proceedings of the CESCG.

[25] Medek P., Beneš P., Sochor J. : Computation of tunnels in protein molecules using Delaunay triangulation. Proceedings of the WSCG.

2009

[26] Zemek M., Kolingerová I. : Hybrid algorithm for deletion of a point in regular and Delaunay triangulation. Spring Conference on Computer Graphics, Budmerice, Slovakia, p. 149-2156.

[27] Zemek M. : Regular triangulation in 3D and its applications. Technical report, University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering, (Internet source: `https://www.kiv.zcu.cz/site/documents/verejne/vyzkum/publikace/technicke-zpravy/2009/tr-2009-03.pdf`).