Marcelo Kallmann

Flexible and Efficient Navigation Meshes for Virtual Worlds

In N. Pelechano, J. Allbeck, M. Kapadia, N. Badler (Eds.), Simulating Heterogeneous Crowds with Interactive Behaviors, 2016. By Marcelo Kallmann. This is the manuscript of the author.

2

Flexible and Efficient Navigation Meshes for Virtual Worlds

CONTENTS

The design and implementation of new techniques for navigation meshes can play a significant role in the navigation capabilities of agents populating modern virtual worlds. This chapter reviews the main approaches used to compute flexible and efficient navigation meshes for 3D virtual worlds, and discusses the use of *Local Clearance Triangulations* as the underlying cell decomposition representing the navigable surfaces in a virtual environment.

2.1 Introduction

Modern virtual worlds tend to be large and rich in details designed to maximize user engagement during interactive experiences. An important part of modeling interactive virtual worlds is the inclusion of its semantic information, which informs multiple modules of the simulation engine controlling the virtual environment. One of the most basic type of semantic information is the definition and representation of the accessible and navigable regions of the environment. This is exactly the purpose of a navigation mesh.

The term navigation mesh [36, 38] has been coined by the computer games community and in general refers to any polygonal mesh that describes navi-

gable surfaces for path planning and other navigation queries. While the term is commonly employed to refer to a structure without any specific underlying construction or property, recent research in the area has greatly contributed to the definition of key approaches for different types of navigation meshes [16].

Choosing a suitable representation for a navigation mesh is important because it will directly influence the types of navigation queries that can be computed, and how efficiently they are computed. A useful navigation mesh has to be flexible to support a number of needed operations, without compromising the ability to compute free paths efficiently. Recent advances have proposed innovative solutions for supporting collision-free paths with arbitrary clearance, real-time dynamic updates, robustness in geometric operations, etc. To address these problems, classical methods from computational geometry and discrete search have been re-visited with new solutions suitable for addressing the real-time constraints of virtual worlds.

The recently introduced Local Clearance Triangulation (LCT) [15] proposes to refine a Constrained Delaunay Triangulation until all narrow passages of the environment can correctly encode clearance information per triangle traversal. The refinements are bounded so that the triangulation remains with $\mathcal{O}(n)$ number of triangles, where n is the number of vertices needed to describe all obstacles in the environment. In this way, an LCT can efficiently answer path queries of arbitrary clearance, allowing the representation to be shared by agents of multiple sizes without the need to compute and maintain the medial axis of the environment (see Figure 2.1). Being a simplicial decomposition, triangulations are flexible to support a number of operations and are often chosen as the starting point for several geometric algorithms relevant to navigation and environment processing.

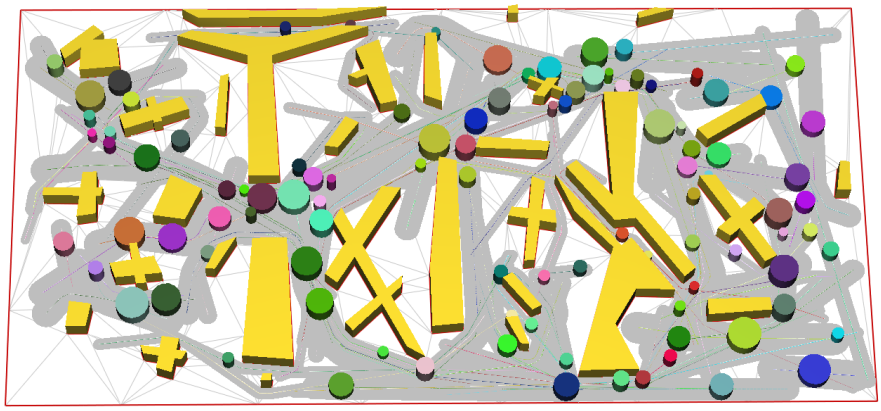


FIGURE 2.1
Local Clearance Triangulation being shared by agents of different clearance requirements. Agents are represented as cylinders. Paths are shown in gray as thick paths delimiting the respective path clearance required by each agent.

This chapter presents an overview of the stages involved in the process of building navigation meshes for virtual worlds, and discusses the use of LCTs as the underlying cell decomposition scheme.

2.2 Computing Navigation Meshes

The main function of a navigation mesh is to represent the navigable regions of a virtual world and to efficiently support the computation of navigation queries. By delimiting the free navigable regions, the navigation mesh also provides important spatial limits that agents can use during collision avoidance and behavior execution. This is a key advantage that navigation meshes offer over graph-based representations such as waypoint graphs or roadmaps.

The process of constructing a navigation mesh can be subdivided in two main phases: detection and extraction of the navigable surfaces in a given 3D virtual world, and cell decomposition and representation of the navigable surfaces. An analysis of these two phases is given in the next subsections. The presented analysis extends an equivalent analysis available in previous work [17].

2.2.1 Detection and Extraction

A number of steps have to be taken into account in order to extract navigable surfaces from a given environment. First of all, the navigation capabilities of the agents to be simulated have to be clearly specified. Based on these capabilities, surfaces that are acceptable for navigation can be then detected and finally connected to each other according to the chosen representations.

Specifying Navigation Capabilities. The most common case involves the specification of limits related to usual human-like locomotion behaviors: the maximum step height that agents can accommodate when climbing stairs or when walking over small obstacles, the maximum terrain slope that agents can accept when navigating on a surface, the minimum height that agents require for being able to pass under obstacles, the maximum jumping distance that agents can overcome with a jumping behavior (when available), etc.

Clearly, depending on the scenario and application at hand, a number of additional parameters can be considered. Different sets of parameters may also be needed in order to specify specific limits for distinct locomotion modes. For example, agents in walking and climbing modes will likely choose different acceptable limits when considering a terrain slope. Agents driving cars or riding bicycles may also be associated with different sets of parameters.

An added complexity appears when a single representation is sought for different types of agents or locomotion modes. For example, if agents can have different sizes, the free regions represented in the navigation mesh have

to include narrow passages that only the smallest agents can traverse. Later at run-time, additional tests will have to be performed for determining if larger agents can pass or not at a given passage. The advantage is that the navigation mesh can be shared by agents of different sizes. This is the case of LCTs [15] and of representations based on the medial axis [8].

In some particular cases the navigation mesh can be optimized for a certain parameter value. For example, if all agents being simulated have the same clearance requirement, the navigation mesh can be already built with boundaries respecting the needed clearance from obstacles. The Recast navigation mesh toolkit [28] offers this capability. In case of agents of different sizes the minimum clearance requirement can still be taken into account in the navigation mesh construction.

When agents can navigate different types of terrains, for example water, grass or pavement, such information should be annotated in the virtual word so that the boundaries between the different types of terrain can be automatically detected and represented. These regions will generally lead to varied traversal costs to be taken into account during path planning, and to locomotion behavior transition points annotated in the navigation mesh. The work of Ninomiya et al. [30] illustrates a number of navigation constraints that can be taken into account, such as avoiding the line of sight of other agents and defining attracting and repelling constraints.

Given the collection of parameters specifying navigation capabilities, the environment can be then processed automatically. Although it is a preprocessing step, fast processing times are important for allowing designers to interactively edit the environment until achieving their design goals. In certain virtual worlds the navigation characteristics of the environment are key to the application. For instance, in strategy and exploration games several regions of the environment are carefully designed with critical navigation goals designed for achieving specific game play experiences. Depending on the goals of the application multiple levels of representation can also be designed in order to account for different types of locomotion behaviors [18].

Processing 3D Worlds. The typical approach is to analyze the virtual world globally with the use of a volumetric decomposition of the whole space occupied by the scene. A volumetric analysis is the most generic approach for handling an environment described without any guarantees on the connectivity or correctness of its polygons. Because it is desirable to not impose any restrictions on the work of designers, the processing has to be robust with respect to degeneracies in the models such as interpenetrating geometry, gaps, etc. The process can also adjust the vertex density describing the boundaries of the navigable surfaces in the scene.

Oliva and Pelechano [33] use GPU techniques to quickly voxelize and process an input scene. The approach first identifies voxels containing scene polygons that respect the navigation capabilities of the agents, and then progressively joins voxels that should make part of a same navigable surface. The result generates multiple navigable layers that are connected to compose

the final navigation mesh. The method used by Recast [28] also relies on a voxelization of the scene. It then partitions the scene with a cell and portal identification method [10] based on the distance field of the voxelized scene.

When the input scene does not require generic volumetric processing, specialized methods operating directly on the input geometry can be developed. For example, Lamarche [22] projects polygons from different layers in the lowest layer to then compute a subdivision that encodes the heights of the layers above it. The information allows to determine navigable regions with respect to height constraints. Later, Jorgensen and Lamarche [14] further subdivide the surfaces according to spatial reasoning metrics able to detect and annotate information relative to rooms and doors.

Building a Unified Representation. Additional steps are required to convert navigable surfaces into a unified navigation mesh representation. At this point semantic information relative to special navigation or access features are considered. For example, doors and elevators will create connections between surfaces, and the connections can be turned on or off at run-time.

An example of a typical special navigation capability that may connect disconnected layers are jumps. A jump can be specified as a simple behavior able to overcome small obstacles, or as a complex behavior that can connect relatively distant layers in varied relative positions. Given a jump specification, layers that can be connected by the jump are typically augmented with special links specifying the connection. These links are usually called off-mesh links, as illustrated in Figure 2.2.

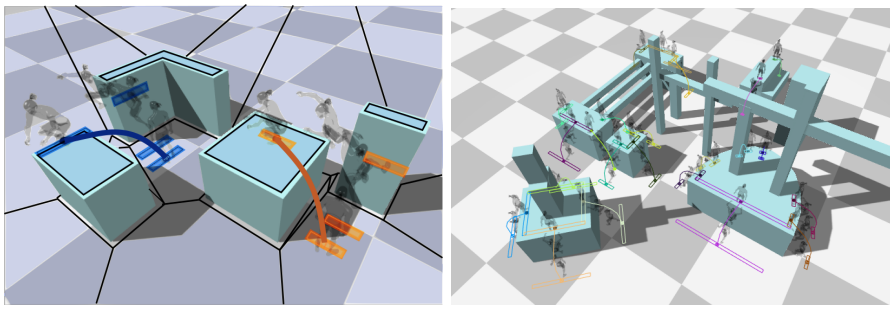


FIGURE 2.2
Example of off–mesh links representing feasible jumps between disconnected surfaces [19].

The final step in the construction of a navigation mesh often involves merging adjacent navigable surfaces, resulting in a multi-layer representation with each layer represented in a chosen polygonal cell decomposition scheme. Most of the approaches are developed as planar decompositions which are extended to connect different layers [39, 33].

2.2.2 Cell Decomposition

The chosen polygonal cell decomposition scheme will play a key role on the properties, efficiency, and types of navigation and path planning queries that can be handled. The analysis that follows is based on selected properties that are important to be observed.

Linear number of cells. A navigation mesh layer should represent the environment with $\mathcal{O}(n)$ number of cells in order to allow search algorithms to operate on the structure at optimal running times. Here n denotes the total number of vertices used to describe the planar polygonal obstacles in the layer.

A linear number of cells will allow the popular Dijkstra [7] and A* [9] graph search algorithms to run on a cell adjacency graph that depends linearly on the number of vertices in the obstacles. Graph search algorithms will then typically run in $\mathcal{O}(n \log n)$ time. This approach is followed by most of the navigation meshes used in practice. Although the search time can be reduced to $\mathcal{O}(n)$ with specialized planar graph search algorithms [21], implementation attempts have not yet been reported in a navigation mesh.

Optimizations are also possible to reduce the number of cells to a minimum. For example, it is possible to build a higher-level adjacency graph connecting only the degree-3 cells, which are the junction cells that connect 3 or more corridors. Such a higher level graph can be encoded in the structure with additional links allowing search algorithms to visit a reduced number of cells. Another optimization is to reduce the number of cells by relying on larger cells. For example, the Neogen approach is based on large almost-convex cells [31]. The drawback is that there is less resolution to encode information or to ensure properties in the mesh.

An important observation is that, while several graph search algorithms will find a globally shortest path in the adjacency graph of a subdivision, a shortest path in the graph will most often not be a globally shortest path in the plane, as discussed next.

Optimality of Computed Paths. Computing globally shortest paths in the plane, or Euclidean shortest paths (ESPs), from a generic cell decomposition is not a simple task. Perhaps the most well-known approach for computing ESPs among polygonal obstacles is to build and search the *visibility graph* [29, 25, 5] of the environment. This can be achieved in $\mathcal{O}(n^2)$ time [34, 37], and although several optimized algorithms exist, this time cannot be reduced for the generic case because the number of edges in the graph is $\mathcal{O}(n^2)$.

Visibility graphs are also difficult to be efficiently maintained in dynamic scenarios. The difficulty comes from the possibly high number of edges and also because visibility is independent of vertex proximity. This leads to local changes often having global effects. Despite these difficulties, visibility graphs still represent the most direct approach for computing shortest paths in the plane.

The ESP problem can however be solved in sub-quadratic time [26] and an

algorithm running in $\mathcal{O}(n \log n)$ time is available [12]. The approach is based on the *continuous Dijkstra* paradigm, which simulates the propagation of a wavefront maintaining equal length to the source point, until the goal point is reached. After the environment is pre-processed in $\mathcal{O}(n \log n)$ time for a given source point, paths to any destination can be retrieved in $\mathcal{O}(\log n)$ time. The pre-processing generates the *Shortest Path Map* (SPM) of the environment, a subdivision of the plane with boundaries being straight line segments or hyperbolic arcs. The approach involves complex geometric computations but GPU techniques recently developed [2] may lead to a practical alternative for achieving optimal paths in applications, in particular when several paths for a same source point are required. See Figure 2.3 for examples. In this case, because the SPM is computed in the frame buffer, a query point can be located in the SPM in constant time and its shortest path to the source will take time proportional to the number of vertices in the path.

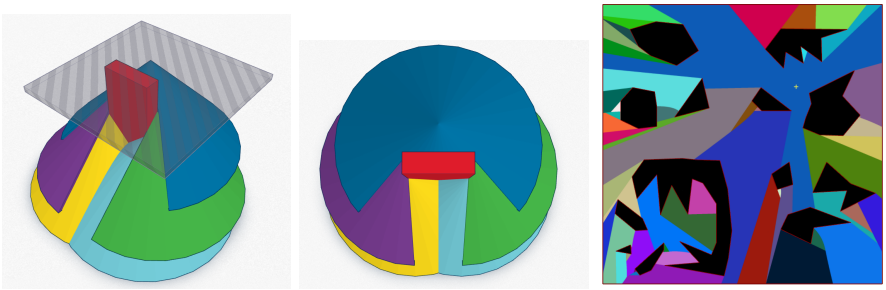


FIGURE 2.3
The shown SPM was computed with GPU rendering [2]. Clipped cones are placed at generator vertices and at heights according to their distances to the source point (left image). Cones are then rendered from an orthographic vertical camera placed above the obstacle plane (center image). The result in the frame buffer will encode the SPM with respect to the source point, which is denoted as a yellow cross (right image). The SPM encodes globally shortest paths to all points in the plane. Given a query point, it is connected to the generator point of the region containing it, then progressively connected to the parent generators until reaching the source node. The traversed sequence of points is the shortest path.

While optimal algorithms for computing ESPs will require specific subdivision structures (like the SPM), triangulations offer a natural approach for cell decomposition and have been explored as the base decomposition for several ESP algorithms. For instance, Kapoor et al. [20] have explored the reduction of a triangulated environment in corridors and junctions in order to compute the relevant subgraph of the visibility graph for a given path query. The method computes globally optimal paths in $O(n + h^2 \log n)$, where h is the number of holes in the planar description of the environment. While other

algorithms for computing ESPs from a triangulation have been explored, the quadratic running time remains a difficult barrier to break.

Although several alternatives exist for computing ESPs, most navigation applications in virtual worlds do not impose the computation of globally shortest paths as a requirement. Fast, simple and robust approaches are often preferred, and the $\mathcal{O}(n \log n)$ path computation time with standard graph search algorithms has been the approach of choice.

A navigation mesh should however facilitate the computation of quality paths. If ESPs cannot always be found, other guarantees on the type of paths that are computed should be provided. A reasonable expectation is that locally shortest paths should be efficiently computed, and additional characterizations related to quality may be adopted. Triangulations, including LCTs, are suitable for computing locally shortest paths efficiently. After a graph search determines a corridor containing a solution path, the shortest path in the corridor can be computed with a linear pass in the triangles of the corridor by using the *funnel algorithm* [3, 24, 11].

Paths with Arbitrary Clearance. Clearance is an important aspect of navigation and a navigation mesh should provide an efficient mechanism for computing paths with arbitrary clearance from obstacles. This means that the structure should not need to know in advance the clearance values that will be used. A weaker and less desirable way to address clearance is to pre-compute information specifically for each clearance value in advance.

The most complete approach for addressing clearance is to explicitly represent the medial axis of the environment [1, 8]. The medial axis can be computed from the Voronoi diagram of the environment, and methods based on hardware acceleration have been developed to improve computation times [13]. One benefit of explicitly representing the medial axis is that locally shortest paths can be easily interpolated toward the medial axis in order to reach maximum clearance when needed. Interpolation toward the maximum clearance path may however not be the most appropriate way of adjusting path clearance and several other approaches are possible. Section 2.4 further discusses this point and presents one alternative approach.

LCTs do not encode the medial axis and instead offer a triangular mesh decomposition that carries just enough clearance information to be able to compute paths of arbitrary clearance, without the need to represent the intricate shapes the medial axis can have. If a path of maximum clearance is required, the medial axis of a triangulated path corridor can still be computed in linear time with available algorithms [4].

Simple techniques for handling clearance directly from a standard Constrained Delaunay Triangulation (CDT) have also been explored, however no simple method has been found to always produce correct results with only local $\mathcal{O}(1)$ time tests. One approach to capture the width of a corridor is to refine constrained edges that have orthogonal projections of vertices from the opposite side of the corridor, adding new free CDT edges with length equal to the width of the corridor [23]. However, such a refinement can only ad-

dress simple corridors and the total number of vertices added to the CDT can be significant. The LCT decomposition provides a solution that correctly and efficiently determines clearance in a triangulation with straight edges. The approach is based on a novel type of refinement operation, and clearance values can be pre-computed and stored in the free edges so that on-line clearance tests are reduced to a simple value comparison per traversed edge. Details are presented in Section 2.3.

Specific pre-computation per clearance value is usually needed when clearance is addressed by structures not specifically designed to capture clearance in all narrow passages of the environment. For example, in the Neogen approach the larger cells require specific computations at the portals for each clearance value to be considered [32].

Representation robustness. A navigation mesh should be robust to degeneracies in the description of the environment. This aspect is first handled during the volumetric extraction of the navigable surfaces in the virtual world (Section 2.2.1), but robustness issues may still arise at the planar level both during construction time and during run-time operation.

It is well-known that the limited precision of floating point operations is often not sufficient for achieving robustness in geometric computations. One approach is to rely on arbitrary precision representation, however imposing a significant performance penalty on the final system. Certain specific operations can be implemented robustly with the use of exact geometric predicates [35, 6].

Robustness becomes particularly difficult when obstacles are allowed to be removed and inserted in the navigation mesh at run-time. When obstacles are inserted undesired self-intersections and overlaps may occur, and intersection points computed with floating point operations may not exactly lie on the intersecting lines. Such imprecision eventually leads to vertices placed at illegal locations. Being robust is crucial for allowing dynamic updates to occur, in particular when users are allowed to make arbitrary updates at run-time.

An approach for handling robust dynamic updates that can be extended to any type of triangulation has been proposed as part of the LCT approach [15]. The solution is based on fast floating point arithmetic and relies on a carefully designed combination of robustness tests, one exact geometric predicate, and adjustment of illegal vertex coordinates. Robustness is achieved for any set of input polygons, including self-intersecting or overlapping polygons, which are robustly handled on-line in any configuration.

Dynamic updates. A navigation mesh should be able to efficiently update itself in order to accommodate dynamic changes in the environment. Dynamic updates are crucial for supporting many common events that happen in virtual worlds. Updates can reflect large changes in the environment or small ones, such as doors opening and closing. An interesting example of small updates is when agents decide to stop for a while and can thus become obstacles for other agents, a situation encountered in specific multi-agent simulations such as in the computer game *The Sims 4* [15].

In general, all approaches for navigation meshes can be extended to accom-

modate dynamic operations. The general trade-off is the more complex the structure is, the more complex and expensive it is to maintain it dynamically. For instance there are several hierarchical representations that are possible to be implemented for speeding up path search; however, if a navigation mesh is associated with a hierarchical structure the hierarchy has also to be updated for every change in the navigation mesh.

The overall chosen approach to address dynamic updates should take into account how often path queries and dynamic updates are executed, and the correct representations and methods should be determined accordingly.

2.3 Local Clearance Triangulations

The properties discussed in the previous section summarize basic needs that navigation meshes should observe in typical virtual world simulations. This section defines Local Clearance Triangulations and later in Section 2.4 the use of LCTs as a flexible and efficient underlying representation for navigation meshes is discussed. A full exposition of LCTs is available in previous work [15].

2.3.1 Definition

Let $\mathcal{S} = \{s_1, s_2, ..., s_m\}$ be a set of m input segments describing polygonal obstacles. Segments in $\mathcal{S}$ may be isolated or may share endpoints forming closed or open polygons. The number of distinct endpoints is n[1] and the set of all endpoints is denoted as $\mathcal{P}$. When inserted in a triangulation, the input segments are also called constraints.

Let T be a triangulation of $\mathcal{P}$, and consider two arbitrary vertices of T to be visible to each other if the segment connecting them does not intercept the interior of any constraint. Triangulation T will be a Constrained Delaunay Triangulation (CDT) of $\mathcal{S}$ if: 1) it enforces the constraints, i.e., all segments of $\mathcal{S}$ are also edges in T, and 2) it respects the *Delaunay criterion* for visible points to each triangle, i.e., the circumcircle of every triangle t of T contains no vertex in its interior which is visible from all three vertices of t.

Although $CDT(\mathcal{S})$ is already able to well represent a given environment, an additional property, the *local clearance property*, is needed in order to achieve correct and efficient clearance determination per triangle during path search.

Let $T = CDT(\mathcal{S})$ and π be a free path in T between points p and q. Path π is considered free if it does not cross any constrained edge of T. A free path may cross several triangles sharing unconstrained edges and the union of all

[1]Here the term *distinct endpoints* is used to clarify that shared endpoints, when existent, should only be considered once when counting the total number of points n.

traversed triangles is called a *channel*. Let t be a triangle in the channel of π such that t is not the first or last triangle in the channel. In this case π will always traverse t by crossing two edges of t. Let a, b, c be the vertices of t and consider that π crosses t by first crossing edge ab and then bc. This particular traversal of t is denoted by τ_{abc}, where ab is the entrance edge and bc is the exit edge. The shared vertex b is called the traversal corner, and the traversal sector is defined as the circle sector between the entrance and exit edges, and of radius $min\{dist(a,b), dist(b,c)\}$, where $dist$ denotes the Euclidean distance. Edge ac is called the interior edge of the traversal. The local clearance of a traversal is now defined.

Definition 1 (Traversal Clearance.) *Given a traversal τ_{abc}, its clearance $cl(a,b,c)$ is the distance between the traversal corner b and the closest vertex or constrained edge intersecting its traversal sector.*

Because of the Delaunay criterion, a and c are the only vertices in the sector, and thus $cl(a,b,c) \leq min\{dist(a,b), dist(b,c)\}$. In case $cl(a,b,c)$ is determined by a constrained edge s crossing the traversal sector, as illustrated in Figure 2.4, then $cl(a,b,c) = dist(b,s)$ and s is the closest constraint to the traversal. If edge ac is constrained, then ac is the closest constraint and $cl(a,b,c) = dist(b,ac)$. If the traversal sector is not crossed by a constrained edge then $cl(a,b,c) = min\{dist(a,b), dist(b,c)\}$.

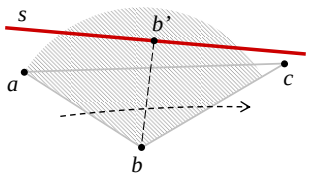


FIGURE 2.4
The triangle traversal with entrance edge ab and exit edge bc is denoted as τ_{abc}. Segment s is the closest constraint crossing the sector of τ_{abc}, thus $cl(a,b,c) = dist(b,s) = dist(b,b')$, where b' is the orthogonal projection of b on s.

The closest constraint to a traversal is now formalized in order to take into account relevant constraints that may not cross the traversal sector of τ_{abc}.

Definition 2 (Closest Constraint.) *Given a traversal τ_{abc}, its closest constraint is the constrained edge s that is closest to the traversal corner b, such that s is either ac or s lies on the opposite side of ac with respect to b.*

In certain situations, the closest constraint of a traversal may generate narrow passages that are not captured by the clearance value of the traversal. The clearance value only accounts for the space occupied by the traversal sector. If a triangle happens to be too thin and long, other vertices not connected

to the traversal may generate narrow passages that are not captured by any clearance value of the involved traversals.

The essence of the problem is that when a triangle is traversed it is not possible to know how the next traversals will take place: if the path will continue in the direction of a possibly long edge (and possibly encounter a narrower space ahead) or if the path will *rotate around* the traversal corner. Each case would require a different clearance value to be considered. For example, Figure 2.7-left shows an example with long CDT triangles where their clearance values are not enough to capture the clearance along the direction of their longest edges. The LCT refinements will fix this problem by detecting these undesired narrow passages and breaking them down into sub-traversals until a single clearance value per traversal can handle all possible narrow passages. The vertices that cause undesired narrow passages are called disturbances, and they are defined below.

Definition 3 (DISTURBANCE.) *Let τ_{abc} be a traversal in T such that its adjacent traversal τ_{bcd} is possible, i.e., edge cd is not constrained. Let s be the closest constraint to τ_{abc} and let v be a vertex on the opposite side of bc with respect to a. Among the vertices connected to v, let d and e be the ones forming $\triangle dve \in T$ crossed by segment vv', where v' is the orthogonal projection of v on s. In this situation, vertex v is a disturbance to traversal τ_{abc} if:*
1. v is not shared by two collinear constraints,
2. v can be orthogonally projected on ac,
3. segment vv' crosses ac and bc,
4. $dist(v, s) < cl(a, b, c)$, and
5. $dist(v, s) < dist(v, e)$.

Figure 2.5 illustrates the definition. A disturbance will always be paired with a constraint disturbing the traversal. A disturbed traversal may contain an arbitrary number of edges between bc and v, however, disturbed traversals will in most cases appear in simpler forms.

Disturbances can occur on any side of a triangle but only need to be defined with respect to the exit edge of a traversal. In this way the set of exit edges for all the possible traversals of a given triangle will address the disturbances that may occur on any traversable side of a triangle. For example, with respect to Figure 2.5, disturbances on the left side of $\triangle abc$ will be detected with respect to τ_{cba}, but not τ_{abc}.

The local clearance triangulation (LCT) can be now defined with the following definitions.

Definition 4 (LOCAL CLEARANCE.) *A traversal τ_{abc} in T has local clearance if it does not have disturbances.*

Definition 5 (LCT.) *A Local Clearance Triangulation is a CDT with all traversals having local clearance.*

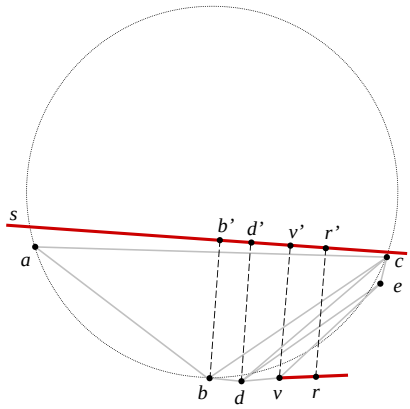


FIGURE 2.5
The shown traversal τ_{abc} is disturbed by vertex v because $dist(v, v') < dist(b, b') = cl(a, b, c)$ and $dist(v, v') < dist(v, e)$. The dashed lines show the orthogonal projections of several vertices on s. Vertices d, e and r are not disturbances since $dist(d, d') > cl(a, b, c)$, $dist(e, s) > dist(e, c)$, and r is shared by two collinear constraints.

2.3.2 Computation

A first approach for computing $LCT(\mathcal{S})$ is based on iterative refinements of disturbed traversals. The algorithm starts with the computation of triangulation $T_0 = CDT(\mathcal{S})$. A linear pass over all traversals of T_0 is then performed, and traversals detected to have a disturbance are refined with one subdivision point p_{ref} added to the current CDT. Every time a constraint $s \in \mathcal{S}$ is refined, s is replaced by two new sub-segments. After all disturbed traversals are processed, a new (refined) set of constraints $\mathcal{S}_1$ is obtained. Triangulation $T_1 = CDT(\mathcal{S}_1)$ is the result of the first global refinement pass. T_1 however may not be free of disturbances and the process has to be repeated until $T_k = CDT(\mathcal{S}_k)$ is free of disturbances, in which case T_k is the desired $LCT(\mathcal{S})$. The number of iterations k mainly depends on the existence of multiple disturbances with respect to a same constraint. The process basically subdivides long edges in order to achieve the local clearance property. Alternatively, the LCT can be built incrementally, maintaining the needed refinements for each segment inserted. In general, incremental operations are more suitable for dynamic updates while global processing of an input CDT is more efficient when computing the LCT for the first time [15].

Let v' be the orthogonal projection of disturbance v on constraint s. A suitable refinement point p_{ref} for solving disturbance v with respect to τ_{abc} and s can be obtained with the mid-point of the intersections of s with the circle passing by vertices d, v and e, where dve is the triangle crossed by segment vv'. See Figure 2.6-left. Most often v will be directly connected to b

and c, and in such case the circle passing by b, v and c is taken. In case of multiple disturbances, v is selected such that no other disturbance on the left side of vv' is closer to s.

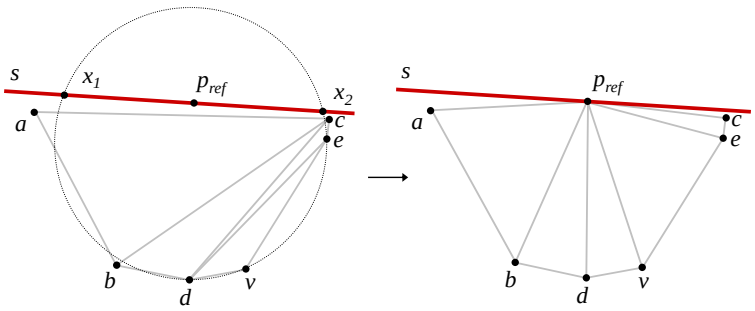


FIGURE 2.6
Vertex v is a disturbance to traversal τ_{abc} and therefore constraint s is subdivided. Points x_1 and x_2 are the intersection points of s and the circle passing by d, v and e. The subdivision point p_{ref} is defined as the midpoint between x_1 and x_2. After refinement, all vertices between b and v will connect to p_{ref}.

Given a desired clearance radius r, the achieved local clearance property guarantees that a simple local clearance test per triangle traversal is enough for determining if a path π can safely traverse a channel with clearance r from constraints. Path π will have enough clearance if $2r < cl(a, b, c)$ for all traversals τ_{abc} of its channel. Figure 2.7 presents an example where local clearance tests are not enough to produce correct results in a CDT, while correct results are obtained in the corresponding LCT.

Lazy Clearance Computation. A lazy approach is used to compute clearance values stored in the edges of the LCT. There are 8 possible traversals passing by an edge, and among them 4 traversals may have distinct values. Each traversal passes by two edges (the entrance and exit edges) and thus only 2 of the 4 values have to be stored per edge.

Clearance values stored in the edges are initialized with a flag (or a negative value) indicating that they have not yet been computed. The values are then computed and stored as needed during path search queries. Every time a path search is launched, each clearance value that is not yet available will be computed and stored in its corresponding edge in order to become readily available for subsequent queries. With this approach, clearance values are only computed in regions reachable by the path queries, avoiding computations in parts of the environment that are not used. The strategy is also valuable during LCT construction and during dynamic updates. Clearance values associated with modified traversals are simply marked as invalid, and later recomputed only when needed by a path query.

Bounded Clearance. One important optimization is to consider the lo-

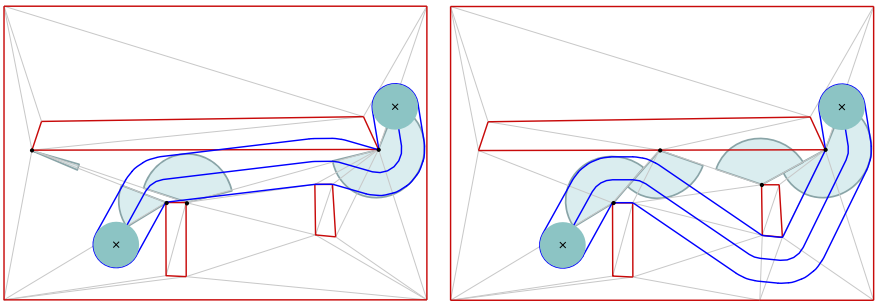


FIGURE 2.7
The left triangulation is a CDT showing an illegal path that however satisfies all its local clearance tests per traversed triangle. The traversal sectors are highlighted and they all have enough clearance. This example shows that local clearance tests per traversal are not enough in CDTs. However, once the existing disturbances are solved and the corresponding LCT is computed (triangulation on the right), local clearance tests become sufficient.

cal clearance property only up to a given maximum value M representing the maximum clearance allowed to be used in path queries. In most cases, M will be the clearance required by the largest agent that needs a path. The triangulation can be then optimized accordingly. Let traversal τ_{abc} be disturbed with respect to disturbance v and constraint s. In order to perform the bounded clearance optimization, refinement operations are adapted to only refine τ_{abc} if $dist(v, s) < min\{cl(a, b, c), M\}$, instead of the original $dist(v, s) < cl(a, b, c)$ condition in Definition 3. This optimization can greatly reduce the number of required refinements, leading to faster computation of the corresponding LCT^M and to less cells processed during path search.

2.3.3 Path Search

Once a LCT of the environment is available, a graph search can be performed over the adjacency graph of the triangulation in order to obtain a channel of arbitrary clearance r connecting two input points p and q. During channel search, a search expansion is only accepted if the clearance of the traversal being expanded (which is precomputed in the free LCT edges) is greater or equal to $2r$.

In addition, LCTs can be safely searched assuming that every cell will be traversed by a given path only once, allowing search algorithms to mark visited triangles and to correctly terminate after visiting each triangle no more than once. Figure 2.8 shows that this is not always the case for all types of cell decompositions.

The example of Figure 2.8 illustrates a situation that is often overlooked

by cell decompositions which are not carefully designed, and that nevertheless has to be addressed in order to guarantee that the employed path search algorithm will correctly execute. A simple proof showing that the situation illustrated in Figure 2.8 cannot happen in CDTs (and in LCTs) is available in previous work [15].

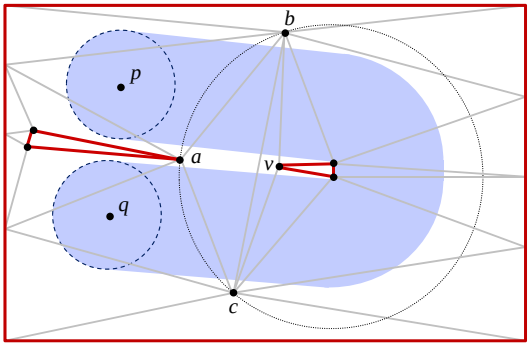


FIGURE 2.8
The shown path is the only solution with clearance r and in the given cell decomposition it traverses $\triangle abc$ and $\triangle bcv$ twice. Clearly, the given triangulation is not a LCT and not a CDT since the circumcircle of $\triangle abc$ has visible vertices in its interior.

2.4 Discussion and Extensions

LCTs address all the expected properties analyzed in Section 2.2.2. Most importantly, although it adds refinements to the underlying CDT, the decomposition remains of linear size. As established in previous work [15], the total number of refinements is limited by the upper bound of $3n$, what translates into a cell decomposition of no more than $6n$ triangles since, using the Euler formula, $t = 2n - 2 - k \Rightarrow t < 2n$, where t is the number of triangles in a triangulation and k is the number of edges in the boundary, i.e., the floor map border ($k = 4$ in all presented examples). In practice, the number of added vertices has shown to be much lower than the bound of $3n$, and the number of triangles remains close to $2n$ [15].

Because the underlying structure is a triangulation, LCTs provide a straightforward solution for computing locally shortest paths. Solution channels are already triangulated and can be quickly processed by the funnel algorithm in order to produce a path that is locally optimal and respecting the desired clearance without the need of any additional data structures or representation conversions. If needed, there are algorithms available for extracting

globally shortest paths directly from a triangulation [20, 27]; however, with either not so simple implementations or with running times worse than $\mathcal{O}(n^2)$.

Clearance information is compactly encoded in LCTs and paths of arbitrary clearance can be efficiently extracted. While LCTs require the maintenance of refinements in the underlying triangulation, it achieves a structure that maintains less nodes than the medial axis [15]. LCTs basically compute refinements just as much as needed in order to determine the maximum clearance (bounded or not) of all its passages. A triangulation is also a simpler structure than the medial axis and algorithms for dynamic updates robust to intersections are available [15].

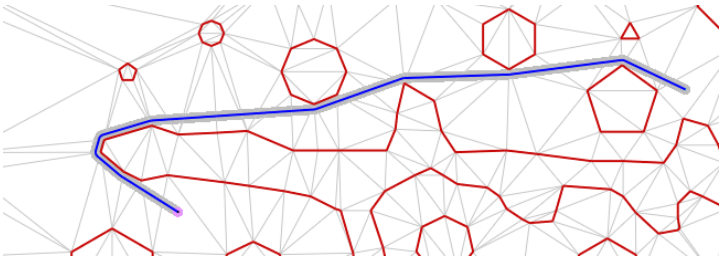
Although the medial axis is not represented, it can be computed in linear time from a channel by using available algorithms [4]. In practice however there is little need for computing exact paths of maximum clearance. A typical path for agents in virtual worlds will be more likely to be one that has a minimum clearance requirement and a desired additional clearance, to be kept when possible, in order to avoid passing too close to obstacles when there is extra space available. Since LCTs encode acceptable clearances for all triangle traversals, path clearance can be adjusted at each traversal as needed. Figure 2.9 exemplifies such a path with customized extra clearance, computed during the funnel algorithm pass by taking into account different clearance values at traversal corners. Additional criteria for further customizing paths can certainly be devised.

Another aspect that is important to be addressed in several virtual world applications is to take into account regions of different types of terrain. In the LCT formulation obstacles are only described by constrained edges, and edges of non-obstacle regions can be equally inserted in the LCT and have their interiors triangulated in the same way as in the free regions of the environment. The insertion algorithms will guarantee that regions defined as closed polygons will remain watertight after insertion, and so flood fill algorithms can be applied to annotate the region type in the interior triangles. In this way, path search can take into account different traversal weights when traversing triangles of different terrain types. Clearance checks during path search have however to be updated to only take into account clearance to obstacle boundaries, since in this case not all constrained edges will be representing obstacles.

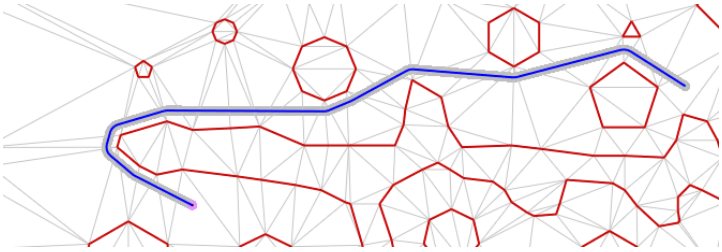
2.5 Conclusion

LCTs introduce a new approach for modeling and computing navigation queries with clearance constraints, and at the same time being able to address key requirements: fast computations, robustness, and dynamic updates. Being a recently developed approach, future work is still needed in order to

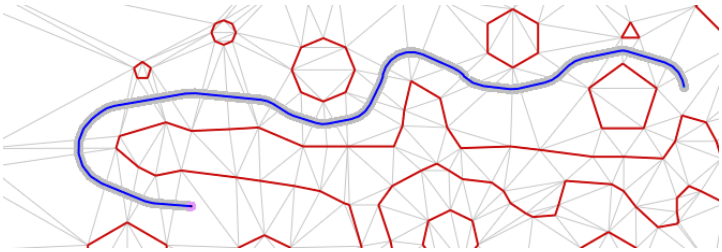
achieve complete solutions integrating LCTs in multi-layered environments and taking into account non-planar surfaces, boundaries of weighted regions, different agent capabilities, etc. Nevertheless, the presented results and possibilities for extensions demonstrate that LCTs achieve a flexible and efficient approach for representing navigation meshes.



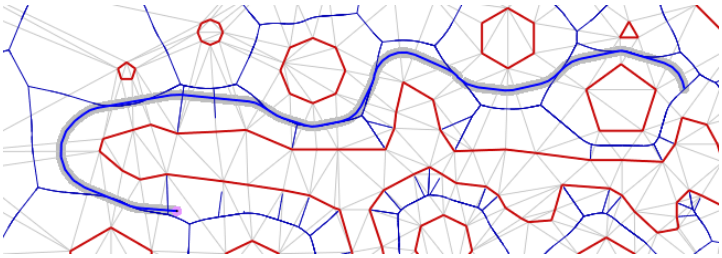
(a) Locally shortest path respecting its minimum clearance requirement.



(b) Path with small extra clearance added at each path corner.



(c) Large extra clearance added, reaching maximum in narrowest passages.



(d) Same path as in the previous case but shown together with the medial axis. Only in the narrowest passages the path converges to the medial axis.

FIGURE 2.9
Different approaches are possible for customizing path clearance. In this example additional clearance is added to a LCT path in order to make it converge towards the medial axis only in the narrowest passages. This is typically an appropriate clearance criterion for paths in virtual worlds.

Bibliography

[1] P. Bhattacharya and M.L. Gavrilova. Roadmap-based path planning - using the voronoi diagram for a clearance-based shortest path. *Robotics Automation Magazine, IEEE*, 15(2):58 –66, june 2008.

[2] Carlo Camporesi and Marcelo Kallmann. Computing shortest path maps with gpu shaders. In *Proceedings of Motion in Games (MIG)*, 2014.

[3] Bernard Chazelle. A theorem on polygon cutting with applications. In *SFCS '82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 339–349. IEEE Computer Society, 1982.

[4] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete and Computational Geometry, In ISAAC: 6th International Symposium on Algorithms and Computation*, 21(3):405–420, 1999.

[5] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational geometry: algorithms and applications*. Springer, 2008.

[6] Olivier Devillers and Sylvain Pion. Efficient exact geometric predicates for delaunay triangulations. In *Proceedings of the 5th Workshop Algorithm Engineering and Experiments*, pages 37–44, 2003.

[7] Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[8] Roland Geraerts. Planning short paths with clearance using explicit corridors. In *ICRA'10: Proceedings of the IEEE International Conference on Robotics and Automation*, 2010.

[9] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[10] D. Haumont, O. Debeir, and F. Sillion. Volumetric cell-and-portal generation. In *Proceedings of EUROGRAPHICS*, 2003.

[11] John Hershberger and Jack Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry Theory and Application*, 4(2):63–97, 1994.

[12] John Hershberger and Subhash Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28:2215–2256, 1997.

[13] Kenneth E. Hoff III, Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. In *ACM Symposium on Computational Geometry*, 2000.

[14] Carl-Johan Jorgensen and Fabrice Lamarche. From geometry to spatial reasoning: automatic structuring of 3D virtual environments. In *Proceedings of the 4th international conference on Motion in Games (MIG)*, pages 353–364, Berlin, Heidelberg, 2011. Springer-Verlag.

[15] Marcelo Kallmann. Dynamic and robust local clearance triangulations. *ACM Transactions on Graphics*, 33(5), 2014.

[16] Marcelo Kallmann and Mubbasir Kapadia. Navigation meshes and real-time dynamic planning for virtual worlds. In *ACM SIGGRAPH 2014 Courses*, SIGGRAPH '14, pages 3:1–3:81, New York, NY, USA, 2014. ACM.

[17] Marcelo Kallmann and Mubbasir Kapadia. *Geometric and Discrete Path Planning for Interactive Virtual Worlds*. Morgan and Claypool Publishers, 2016.

[18] Mubbasir Kapadia, Alejandro Beacco, Francisco Garcia, Vivek Reddy, Nuria Pelechano, and Norman I. Badler. Multi-domain real-time planning in dynamic environments. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '13, pages 115–124, New York, NY, USA, 2013. ACM.

[19] Mubbasir Kapadia, Xu Xianghao, Marcelo Kallmann, Maurizio Nitti, Stelian Coros, Robert W. Sumner, and Markus Gross. PRECISION: Precomputed Environment Semantics for Contact-Rich Character Animation. In *Proceedings of the 2016 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D'16, New York, NY, USA, 2016. ACM.

[20] Sanjiv Kapoor, S. N. Maheshwari, and Joseph S. B. Mitchell. An efficient algorithm for euclidean shortest paths among polygonal obstacles in the plane. In *Discrete and Computational Geometry*, volume 18, pages 377–383, 1997.

[21] Philip Klein, Satish Rao, Monika Rauch, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. In *Journal of Computer and System Sciences*, pages 27–37, 1994.

[22] Fabrice Lamarche. Topoplan: a topological path planner for real time human navigation under floor and ceiling constraints. *Computer Graphics Forum*, 28(2):649–658, 2009.

[23] Fabrice Lamarche and Stephane Donikian. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, 23(3):509–518, 2004.

[24] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 3(14):393–410, 1984.

[25] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of ACM*, 22(10):560–570, 1979.

[26] Joseph S. B. Mitchell. Shortest paths among obstacles in the plane. In *Proceedings of the ninth annual symposium on computational geometry (SoCG)*, pages 308–317, New York, NY, USA, 1993. ACM.

[27] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, August 1987.

[28] Mikko Mononen. Recast navigation mesh toolset, 2015. `https://github.com/memononen/recastnavigation`.

[29] N. Nilsson. A mobile automaton: an application of artificial intelligence techniques. In *In Proceedings of the 1969 International Joint Conference on Artificial Intelligence (IJCAI)*, pages 509–520, 1969.

[30] Kai Ninomiya, Mubbasir Kapadia, Alexander Shoulson, Francisco Garcia, and Norman Badler. Planning approaches to constraint-aware navigation in dynamic environments. *Computer Animation and Virtual Worlds*, 2014.

[31] R. Oliva and N. Pelechano. Automatic generation of suboptimal navmeshes. In *In Proceedings of the Fourth International Conference on Motion in Games (MIG)*, 2011.

[32] R. Oliva and N. Pelechano. A generalized exact arbitrary clearance technique for navigation meshes. In *In Proceedings of the ACM SIGGRAPH conference on Motion in Games (MIG)*, 2013.

[33] R. Oliva and N. Pelechano. Neogen: Near optimal generator of navigation meshes for 3D multi-layered environments. *Computer & Graphics*, 37(5):403–412, 2013.

[34] M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proceedings of the fourth annual symposium on Computational geometry (SoCG)*, pages 164–171. ACM, 1988.

[35] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3):305–363, October 1997.

[36] Greg Snook. Simplified 3d movement and pathfinding using navigation meshes. In Mark DeLoura, editor, *Game Programming Gems*, pages 288–304. Charles River Media, 2000.

[37] James A. Storer and John H. Reif. Shortest paths in the plane with polygonal obstacles. *J. ACM*, 41(5):982–1012, 1994.

[38] Paul Tozour. Building a near-optimal navigation mesh. In Steve Rabin, editor, *AI Game Programming Wisdom*, pages 171–185. Charles River Media, 2002.

[39] Wouter G. van Toll, Atlas F. Cook IV, and Roland Geraerts. Navigation meshes for realistic multi-layered environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3526–3532, 2011.