The Medial Axis of a Multi-Layered Environment and Its Application as a Navigation Mesh

WOUTER VAN TOLL, Utrecht University
ATLAS F. COOK IV, University of Hawaii at Manoa
MARC J. VAN KREVELD, Utrecht University
ROLAND GERAERTS, Utrecht University

Path planning for walking characters in complicated virtual environments is a fundamental task in simulations and games. A *navigation mesh* is a data structure that allows efficient path planning. The Explicit Corridor Map (ECM) is a navigation mesh based on the *medial axis*. It enables path planning for disk-shaped characters of any radius.

In this article, we formally extend the medial axis (and therefore the ECM) to 3D environments in which characters are constrained to walkable surfaces. Typical examples of such environments are multi-storey buildings, train stations, and sports stadiums. We give improved definitions of a *walkable environment* (WE: a description of walkable surfaces in 3D) and a *multi-layered environment* (MLE: a subdivision of a WE into connected layers). We define the medial axis of such environments based on projected distances on the ground plane. For an MLE with n boundary vertices and k connections, we show that the medial axis has size $O(n)$, and we present an improved algorithm that constructs the medial axis in $O(n \log n \log k)$ time.

The medial axis can be annotated with nearest-obstacle information to obtain the ECM navigation mesh. Our implementations show that the ECM can be computed efficiently for large 2D and multi-layered environments and that it can be used to compute paths within milliseconds. This enables simulations of large virtual crowds of heterogeneous characters in real-time.

CCS Concepts: • **Theory of computation** → **Computational geometry**; • **Computing methodologies** → *Mesh geometry models*; *Modeling and simulation*; • **Mathematics of computing** → *Geometric topology*;

Additional Key Words and Phrases: Medial axis, Voronoi diagram, multi-layered environment, navigation mesh, path planning, crowd simulation

ACM Reference format:
Wouter van Toll, Atlas F. Cook IV, Marc J. van Kreveld, and Roland Geraerts. 2018. The Medial Axis of a Multi-Layered Environment and Its Application as a Navigation Mesh. *ACM Trans. Spatial Algorithms Syst.* 4, 1, Article 2 (May 2018), 34 pages.
https://doi.org/10.1145/3204456

This research was supported by the COMMIT project (http://www.commit-nl.nl/).
Authors' addresses: W. van Toll, M. J. van Kreveld, and R. Geraerts, Utrecht University, Department of Information and Computing Sciences, Princetonplein 5, 3584 CC, Utrecht, Netherlands; emails: {W.G.vanToll, M.J.vanKreveld, R.J.Geraerts}@uu.nl; A. F. Cook IV, University of Hawaii at Manoa, 1680 East-West Road, Honolulu, HI 96822, USA; email: acook4@hawaii.edu.

https://doi.org/10.1145/3204456

1 INTRODUCTION

In many simulations and gaming applications, virtual characters need to plan and traverse visually convincing paths through a complicated environment in real time. We focus on entities that move along walkable surfaces; we will refer to these entities as *characters*. Characters should move smoothly and avoid collisions with obstacles and other characters. The environment in which they move is typically 3D, but characters are constrained to walkable surfaces. These surfaces form the *walkable environment* (WE). It is often useful to subdivide the WE into planar layers. We refer to such a subdivision as a *multi-layered environment* (MLE).

A *navigation mesh* is a subdivision of the WE into polygons for the purpose of path planning. The *dual graph* of this mesh has a vertex for each polygon in the mesh and an edge for each pair of adjacent polygons. Characters can search in this graph to find *global* routes, which they traverse while *locally* avoiding other characters.

In this article, we work toward a refined definition and construction algorithm of the Explicit Corridor Map (ECM) [14, 62]. The ECM is a navigation mesh based on the medial axis. The medial axis of an environment is the set of all points in the environment with more than one closest obstacle; we will define this more precisely in Section 3 (for 2D environments) and Section 5.1 (for WEs).

First, we revisit the medial axis in 2D. Next, we give improved definitions of WEs and MLEs, and we define the medial axis of a WE and MLE based on projected distances onto the ground plane. For an MLE with n boundary vertices and k connections between layers, where each connection is a line segment when projected onto the ground plane, we show how to compute the medial axis in $O(n \log n \log k)$ time. Our algorithm uses several non-trivial insights into the characteristics of a WE.

The medial axis of an MLE can easily be annotated to obtain the ECM navigation mesh. Figure 1 shows the ECM for a simple MLE. The ECM can be used to plan paths for disk-shaped characters of any radius, which is typically not possible when using an arbitrary subdivision into polygons. Because the ECM is a sparse graph that uses only $O(n)$ storage, it is suitable for efficient path planning. It also supports various operations that are important for crowd simulation, such as finding the nearest static obstacle to a query point. Furthermore, it can be updated in real time when obstacles appear or disappear. Combined with algorithms for path following and collision avoidance, the ECM can be used to simulate large crowds of heterogeneous characters in real time.

1.1 Contributions

This article formalizes and extends previous conference publications [14, 62] by defining and proving the essential characteristics of the medial axis and ECM in MLEs. Compared to our previous work, the main *contributions* of this article are the following:

- We give improved definitions of WEs, MLEs, and the medial axis of a WE or MLE based on projected distances (Sections 4 and 5.1).
- We solve a problem in our previous construction algorithm for the medial axis of an MLE [62]. We present an improved algorithm, and we prove that this algorithm is correct and that it runs in $O(n \log n \log k)$ time (Section 5).
- We give a refined definition of the ECM navigation mesh (Section 6).
- We show via experiments that our implementation can efficiently generate the ECM and compute paths in large MLEs (Section 8).

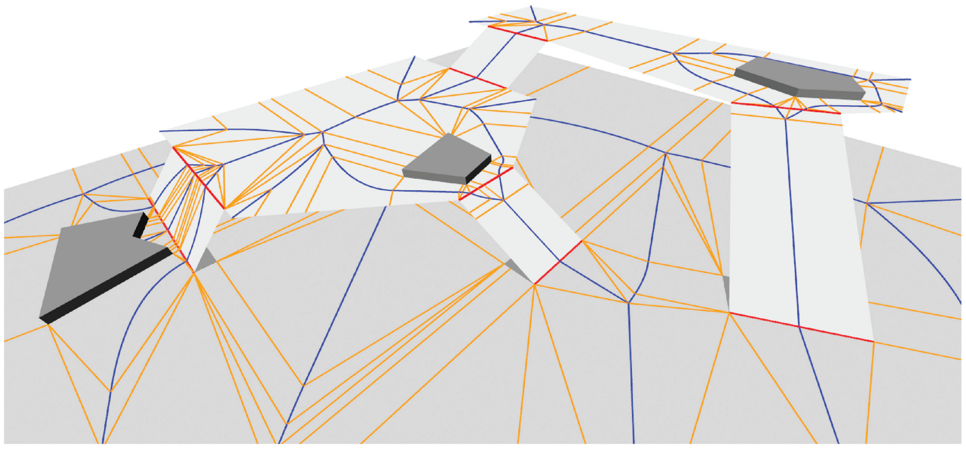


Fig. 1. 3D view of an MLE and its ECM navigation mesh. The ECM is a medial axis (the dark graph) annotated with nearest-obstacle information (the light line segments).

2 RELATED WORK ON PATH PLANNING AND NAVIGATION MESHES

This section summarizes related research on path planning, navigation meshes, and crowd simulation. Several good overview books of this research area exist [28, 61]. We will focus on the topics that are particularly relevant to the ECM navigation mesh. Related work on its underlying structure, the medial axis, will be covered in Section 3.

2.1 Path Planning

In traditional motion planning, a robot needs to compute a collision-free trajectory from one configuration to another [37, 38]. The number of degrees of freedom for the robot determines the dimensionality of the *configuration space*. For high-dimensional spaces, exact solutions are typically intractible and *probabilistic* methods are often successful [32, 36]. Such methods do not compute the configuration space explicitly. Instead, they represent it using a graph of sampled collision-free configurations and motions. Several of these techniques generate samples that lie on the Voronoi diagram or medial axis [12, 25, 42]. The *exact* medial axis is too difficult to compute whenever there are three or more dimensions [2], which justifies the use of sampling by these techniques. However, in this article, we do not look at high-dimensional motion planning, but at path planning for disks in MLEs. We will show how to compute the medial axis of an MLE in an exact manner based on projected distances.

In crowd simulation, characters are typically modeled as disks that move along walkable surfaces, which enables different types of algorithms than in traditional motion planning. Path planning for disks can be solved by inflating the obstacles in the environment by the character's radius (i.e., by computing Minkowski sums) and then computing a path for a point character [37, 38]. This approach requires a separate inflation process for each distinct radius.

To plan a shortest path for a point in a 2D space, one can use a shortest-path map [20] or a visibility graph, which has $O(n^2)$ edges for an environment with n obstacle vertices [15]. The Visibility-Voronoi Complex [69] is an extension that implicitly encodes the visibility graph for all disk sizes; it can generate the visibility graph for a particular radius on the fly.

Path planning for large crowds typically uses a less complex graph (or *roadmap*) that does not always yield the shortest path. The medial axis is such a roadmap; its application to path planning for disk-shaped characters is referred to as the "retraction method" [46], and we use it as a basis for our ECM navigation mesh. Roadmaps have been used frequently for crowd simulation [23, 57]. Characters should be allowed to locally deviate from the graph's edges to increase variety in the crowd and to avoid unnatural motions and collisions between characters. However, if the graph does not store information about where the obstacles are, then it is relatively expensive to compute these deviations. In a navigation mesh (which we will discuss in the next subsection), such computations are easier, because the graph and the obstacle representation are united.

Another popular approach to path planning is to use a *grid* that subdivides the environment into regular cells. Grids are easy to implement and well studied by the path planning community (see, e.g., References [13, 35, 41, 60]). However, grids have resolution problems: A coarse grid (with few cells) does not capture the environment's details, whereas a fine grid (with many cells) quickly becomes too costly to store and query.

2.2 Navigation Meshes in 2D Environments

A *navigation mesh* subdivides the configuration space into polygonal regions [59]. A global path in a navigation mesh can be found by performing A* search [17] on the dual graph of the mesh. The result is a sequence of regions to move through, such that characters can use the available space to locally adjust their movements during the simulation. This makes navigation meshes more flexible for crowd simulation than standard graphs. Navigation meshes are also more scalable to large environments than grids [66].

Many navigation meshes exist for 2D environments; they are typically *exact* subdivisions of the configuration space. Examples are the Local Clearance Triangulation (LCT) [27] and the Explicit Corridor Map (ECM) [14]. These navigation meshes have the advantage that they encode *clearance* information. As such, they can be used to compute paths for disk-shaped characters with an arbitrary radius without explicitly inflating any obstacles.

2.3 Navigation Meshes in 3D Environments

Navigation meshes in 3D are usually designed for environments with a consistent direction of gravity. We will use the term *walkable environment* to denote the surfaces on which characters can walk. Many navigation mesh techniques automatically convert a 3D environment to a WE by discretizing the environment into traversable and non-traversable cubes, or *voxels*. The pioneering work by Pettré et al. [51] essentially computes an *approximation* of the multi-layered medial axis that we define in this article. Their navigation mesh supports arbitrary character sizes but uses overlapping disks that do not completely cover the environment. The Recast Navigation toolkit [43] is very popular in the computer games industry, e.g., it is included in the Unity game engine [68]. Oliva and Pelechano have presented a similar method called NEOGEN [49], and they have investigated path planning for disks in arbitrary navigation meshes [48]. A disadvantage of voxels is that they do not scale well to large environments, because these environments require many voxels to obtain sufficient precision. Therefore, exact alternatives to 3D filtering are also being investigated [53].

For many applications, it is useful to subdivide the WE into layers such that each layer can be treated as a planar problem space. We will refer to such a subdivision as an *MLE*. In this article, we extend the medial axis and the Explicit Corridor Map to MLEs. There are several ways to obtain an MLE from a WE. Deusdado et al. have used rendering techniques assuming certain properties such as axis-alignment [9], and Whiting et al. have shown how to extract layers from a CAD drawing [70]. For an arbitrary WE represented by a triangle mesh, Hillebrand [21] has proven that obtaining an optimal MLE (with a minimum number of connections) is NP-hard in the number of triangles, but he has shown that good results can be obtained using heuristics [22].

A consequence of splitting a WE into layers and treating each layer as a 2D space is that *height differences* along the surface are ignored. In all navigation meshes based on this principle (including ours), slopes are not considered to affect the length of a path, and path lengths are effectively projected onto the ground plane. This can be seen as a disadvantage.

However, finding short paths on terrains with height differences is known to be substantially more difficult [31]. Recently, researchers have suggested to use the 3D environment itself as a navigation data structure [5, 56], which coincidentally also supports environments with arbitrary gravity directions. At the time of writing, these types of solutions are less mature; for instance, it is unclear how to compute paths with arbitrary clearance and how to properly extend collision-avoidance algorithms to this domain.

By contrast, *projected distances* provide a simplification that allow solutions in 2D to be extended to 3D environments with a consistent gravity direction. In this article, we will provide this extension for the medial axis and the ECM. Extending these concepts further to also encode height differences is a topic for future work.

2.4 Comparing Navigation Meshes

We have recently conducted a comparative study of navigation meshes [66], using unified definitions, objective quality metrics, and a benchmark suite that runs on a single computer. This comparison included the multi-layered ECM from this article, several other state-of-the-art navigation meshes [27, 43, 49, 51], and a simple grid-based method. Our study confirmed that voxel-based extraction of a WE is not very scalable to large environments, which justifies a search for alternatives.

For an analysis of the theoretical and practical (dis)advantages of the ECM compared to other navigation meshes, we refer the reader to this comparative study [66]. We will not repeat the comparison here; instead, we will focus on definitions, algorithms, proofs, and experiments specifically relevant to the medial axis and ECM.

2.5 Crowd Simulation

Navigation meshes are useful for simulating crowds of virtual characters with individual properties and goals. Path planning on a navigation mesh leads to an *indicative route* that can be traversed smoothly in real time [26]. Each character can locally avoid collisions with other characters using forces [18, 55] or velocity selection [4, 29, 44]. Many other algorithms exist that can improve global coordination and local behavior. These components can be mixed arbitrarily in a multi-level crowd simulation framework [65].

Other crowd simulation algorithms aim to unify the global and local planning levels by defining a *potential field*: a grid representation of the environment that stores the optimal walking direction (toward a particular goal region) in each cell. These directions are updated in real time in response to the crowd's movement. While potential fields in general can contain local minima in which characters would get stuck, various solutions specific to crowd simulation have alleviated this problem via global optimization [45, 50, 67]. Potential fields can efficiently model dense crowds with many characters sharing the same goals and properties. However, because each goal region and behavior type requires its own potential field, these methods do not allow for large *heterogeneous* crowds in which each character has different properties. If individuality is required, then navigation meshes with local collision avoidance are preferred.

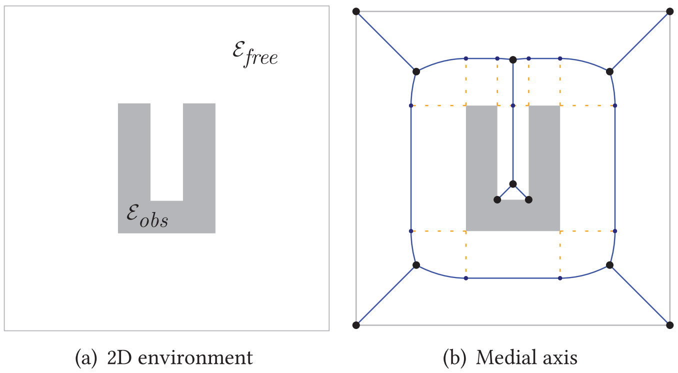


Fig. 2. A simple 2D environment and its medial axis. (a) The obstacle space $\mathcal{E}_{obs}$ (shown in gray) consists of line segments and polygons. Its complement is the free space $\mathcal{E}_{free}$. (b) The medial axis is a graph through $\mathcal{E}_{free}$. True vertices are shown as large dots. Semi-vertices (small dots) occur when a bisector's generator changes. This is indicated by dashed segments, which are not part of the graph.

3 PRELIMINARIES: THE MEDIAL AXIS IN 2D ENVIRONMENTS

In this section, we give a formal definition of a 2D virtual environment and its medial axis. These concepts are not novel, but they are required for understanding the extension to WEs and MLEs that we will present next.

3.1 2D Environment

Let $\mathcal{E}$ be a bounded 2D planar environment with polygonal obstacles. We define the *obstacle space* $\mathcal{E}_{obs}$ as the union of all obstacles, including the boundary of the environment. The complement of $\mathcal{E}_{obs}$ is the *free space* $\mathcal{E}_{free}$. An example of such an environment is shown in Figure 2(a). Let n be the number of vertices that define the boundary of $\mathcal{E}_{free}$ using interior-disjoint simple polygons, line segments, and points. We call n the *complexity* of $\mathcal{E}$.

3.2 Medial Axis

The medial axis is a variant of the *Voronoi diagram* (VD) [2, 47]. For a planar set of point sites, the VD is a subdivision of the plane into cells such that all points in a cell have the same nearest site. The *edges* of the VD are parts of bisectors: line segments or half-lines on which every point is equidistant to two sites. These bisectors meet at *vertices* that are equidistant to at least three sites.

The VD can be extended to handle line segments and polygons as sites. This version is sometimes called the *generalized Voronoi diagram* or GVD [14, 40]. The edges of a GVD consist of line segments and parabolic arcs, and degree-2 vertices occur at the positions where a bisector changes its shape. Several robust implementations of the GVD exist [7, 19, 24, 30]. There are multiple definitions of the GVD, differing mostly in how they handle site vertices shared by multiple sites. The term *generalized Voronoi diagram* is also used for other generalizations of the VD [47].

The *medial axis* has been extensively studied in the field of computational geometry, usually for 2D polygons with or without holes [6, 8, 39, 54, 71]. As mentioned in Section 2.1, the medial axis is also used frequently for motion planning in high-dimensional configuration spaces. In such spaces, the medial axis is too difficult to compute exactly, so it needs to be approximated via sampling techniques. We will now focus on the 2D domain.

Informally, the medial axis of a polygon P can be seen as the GVD of P's boundary segments, restricted to (i.e., intersected with) the interior of P. Several definitions of the medial axis exist;

they mainly differ in their choice of which edges to prune from the GVD. We therefore give our own definition, which is comparable to those by Preparata and Lee [39, 54], but applied specifically to a 2D environment and its free space.

Definition 3.1 (Medial axis, 2D). For a bounded 2D environment $\mathcal{E}$ with closed polygonal obstacles, let $ma(\mathcal{E})$ be the set of all points in $\mathcal{E}_{free}$ that have at least two distinct equidistant nearest points on the boundary of $\mathcal{E}_{free}$, in terms of 2D Euclidean distance. The medial axis $MA(\mathcal{E})$ is the topological closure of $ma(\mathcal{E})$.

Figure 2(b) shows the medial axis of an example environment. Since the medial axis is a pruned Voronoi diagram, it forms a plane graph (a planar graph embedded in 2D). The term *closure* ensures that degree-1 medial axis vertices (e.g., at the corners of the bounding box) are also included. Note that the medial axis does not run into obstacle corners with an angle of $\leq$180 degrees (*convex* corners), such as most corners of the U shape in Figure 2(b). Such edges *do* appear in a GVD of line segment sites, because these corners are then shared by multiple sites.

Each medial axis arc A is the bisector of two *generators*: the endpoints or segments of $\mathcal{E}_{obs}$ that are nearest to A. If one generator is a line segment and the other is a point, then A is a parabolic arc; otherwise, A is a line segment.

In this article, we refer to all medial axis vertices of degree 1, 3, or higher as *true vertices*. We refer to the degree-2 vertices as *semi-vertices*, because the medial axis only changes its shape at these points. Observe from Figure 2(b) that a semi-vertex occurs when the medial axis crosses a normal vector at a convex obstacle corner. We define an *edge* as a sequence of medial axis arcs between two true vertices.

3.3 Complexity of the Medial Axis

It is well-known that the GVD of non-crossing line segments with n distinct endpoints has $O(n)$ vertices and edges and can be computed in $O(n \log n)$ time [2, 3]. The three most common construction algorithms for the point-site Voronoi diagram (plane sweep [11], randomized incremental construction [16], and divide-and-conquer [58]) can each be extended to support line-segment sites as well [2].

The medial axis has the same asymptotic size of $O(n)$, because it is a pruned GVD. The medial axis can be obtained from the GVD without increasing the overall asymptotic running time [8, 34].

4 DEFINITIONS OF WALKABLE AND MULTI-LAYERED ENVIRONMENTS

In this section, we define the types of environments *embedded in 3D* for which we want to construct a navigation mesh. Our main assumption is that there is a consistent direction of gravity throughout an environment. For example, we support multi-storey buildings, but not spherical planets. As explained in Section 2, this is a common assumption for many navigation meshes, and we consider other 3D surfaces to be outside the scope of this article.

Throughout this article, a *3D environment* is a collection of polygons in $\mathbb{R}^3$. These polygons may include floors, ceilings, walls, or any other type of geometry. Figure 3(a) shows a simple example of a 3D environment.

4.1 Walkable Environment

Informally, a WE can be thought of as a set of polygonal surfaces in $\mathbb{R}^3$ on which characters can walk. Characters can move from one polygon onto another if these polygons are connected in 3D. The *free space* $\mathcal{E}_{free}$ of a WE is simply the entire set of surfaces. Unlike in 2D environments, the obstacle space $\mathcal{E}_{obs}$ is not intuitively defined, but we will sometimes refer to points on the boundary of $\mathcal{E}_{free}$ as *obstacle points*. Figure 3(b) shows a simple example of a WE.

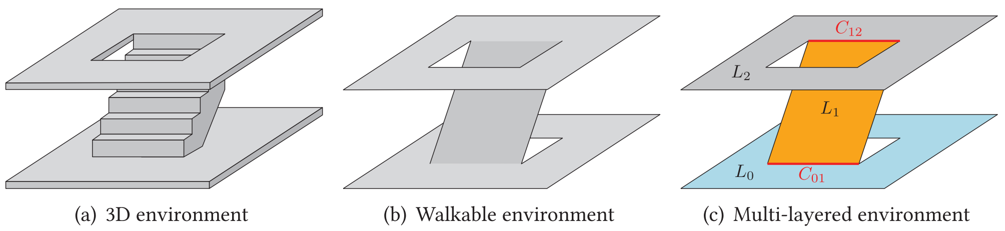


Fig. 3. A simple 3D environment for which we want to compute a navigation mesh. (a) The original environment is a collection of polygons in 3D. (b) A WE is a set of polygons along which characters can walk. (c) An MLE is a subdivision of the WE into 2D layers. Connections between layers are shown as bold line segments in this example.

A WE may consist of multiple connected components: For example, consider two islands with no bridge connecting them. In topological terms, each component is an orientable 2-manifold (a *surface*) with a boundary. This intuitively means that the WE has a "top" and "bottom" side, and any point on the bottom side cannot be reached from a point on the top side without intersecting a boundary. Geometrically, we are only interested in the top side, i.e., the floors and not the ceilings. The WE is also what we call *direction-consistent*: Slopes are allowed, but there is a single gravity direction for the entire environment. This leads to the following formal definition:

Definition 4.1 (Walkable environment). A walkable environment (WE) is a set of interior-disjoint polygons in $\mathbb{R}^3$. Topologically, each connected component of a WE is an orientable 2-manifold with a boundary. Geometrically, the WE is direction-consistent: There exists a horizontal ground plane P below the WE such that for any non-boundary point q, the infinitesimal neighborhood $\sigma(q)$ of q does not overlap itself when projected vertically down onto P. Semantically, the WE represents all polygons on which characters can walk.

In theory, it does not matter how a WE is created. In practice, a WE can be obtained from a 3D environment by filtering out unwalkable parts, e.g., surfaces that are too steep and surfaces along which the ceiling is too low for characters to pass under. (Note that filtering out steep surfaces leads to direction-consistent output.) Such a filtering process typically also merges polygons that are nearly adjacent; for example, staircases are converted into ramps. As explained in Section 2.3, the details of these filtering techniques are outside the scope of this article. Voxel-based implementations exist [43, 49, 51], and exact alternatives are in development [53].

Once more, it is important to note that the entire WE can be self-overlapping when projected onto the ground plane P. This is the main difference to 2D environments, and it is the main reason that we introduce MLEs next.

4.2 Multi-Layered Environment

An MLE is a subdivision of a WE into *layers* such that each individual layer can be projected onto the ground plane P without overlap. Although a single layer does not need to have a particular meaning, a typical example of a layer is one floor of a building.

A subdivision into layers is useful for many purposes, including visualization (each layer can be drawn in 2D), identification (all surface points can be uniquely specified using a 2D position and a layer ID), and the construction of geometric data structures (existing 2D construction algorithms can be applied to each layer). In Section 5, we will use the concept of layers to compute the medial axis of a WE.

The layers of an MLE are connected by curves that we call *connections*. Intuitively, they are the "cuts" that were introduced during the subdivision into layers, and they are the edges along which the layers can be "glued together" to obtain the original WE. To facilitate the algorithm of Section 5, we require that connections have particular geometric properties. Formally, we define an MLE as follows:

Definition 4.2 (Multi-layered environment). A multi-layered environment (MLE) is a WE that has been subdivided into l planar layers, $\mathcal{L} = \{L_0, \ldots, L_{l-1}\}$, using a set $C = \{C_0, \ldots, C_{k-1}\}$ of k connections.

Each layer $L_i \in \mathcal{L}$ is a set of walkable surfaces that are non-overlapping when projected onto the ground plane P. The free space $\mathcal{E}_{free,i}$ of L_i is the union of all polygons in L_i. Combining the free space of all layers yields the free space $\mathcal{E}_{free}$ of the original WE.

Each connection $C_q \in C$ is a curve with the following properties:

- It lies on the shared boundary of two layers L_i and L_j ($i \neq j$), thus connecting the walkable polygons of these layers.
- Its endpoints lie on existing boundary vertices of $\mathcal{E}_{free}$, so its endpoints are impassable obstacles.
- Its interior lies entirely inside the interior of $\mathcal{E}_{free}$, so it follows the surface of the WE without intersecting the boundary of $\mathcal{E}_{free}$.
- Its interior does not intersect any other connections.
- Its projection onto the ground plane P is a straight line segment.

Figure 3(c) shows an example of an MLE. Note that the MLE is still *embedded in 3D*, but each individual layer *can* be projected onto P without self-overlap, if desired. Therefore, the projection of a layer L_i onto P is essentially a 2D environment with obstacles as described in Section 3.1. The boundary vertices of these obstacles are also boundary vertices of $\mathcal{E}_{free}$. (Conversely, if we embed a 2D environment in $\mathbb{R}^3$, we obtain a special case of a WE, which is also an MLE with one layer and no connections.)

The connections in the example of Figure 3(c) are straight line segments in $\mathbb{R}^3$, but this is not necessary in general: A connection can be any curve that satisfies the constraints given in Theorem 4.2. These constraints are important for our construction algorithm in Section 5.

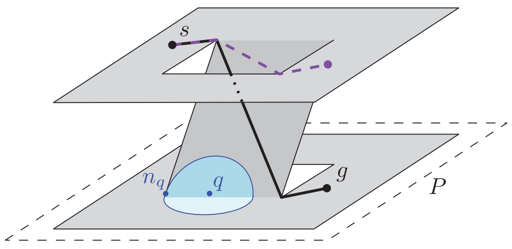
Two layers L_i and L_j may be connected through multiple connections at different positions; for example, imagine a bridge that connects to the same layer at both ends. Also, a subdivision into layers is usually not unique: Any subdivision that meets the requirements described above is acceptable. As described in Section 2.3, obtaining an MLE with a minimum number of connections is NP-hard [21], but there are several heuristic approaches to obtaining a valid MLE.

4.3 Complexity of a Multi-Layered Environment

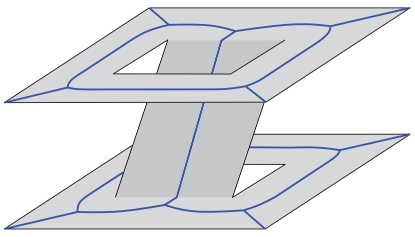
The complexity of an MLE is given by the number of connections k and the number of obstacle vertices n in all layers combined. Let n_i be the number of obstacle vertices in a layer L_i. We define n as $\sum_{i=0}^{l-1} n_i$. Note that a vertex occurs in multiple layers if it is an endpoint of a connection. The following lemma bounds the number of connections.

Lemma 4.3. *For any MLE with l layers and n obstacle vertices, the number of connections k is $O(n)$.*

Proof. Let n_i be the number of obstacle vertices in a layer L_i. By definition, $n = \sum_{i=0}^{l-1} n_i$. In each layer L_i, the number of connections is bounded by the maximum number of non-intersecting line segments that can be drawn between its n_i vertices. Euler's formula for planar graphs implies that this is $O(n_i)$. Therefore, the total number of connections is $O(\sum_{i=0}^{l-1} n_i) = O(n)$. □



(a) Distances



(b) Medial axis

Fig. 4. The medial axis of a WE $\mathcal{E}$ is based on path lengths projected onto the ground plane P. (a) The shortest path between two points s and g is shown as a bold curve. Its projected length is indicated by the dashed curve. For a non-boundary point $q \in \mathcal{E}_{free}$ with a nearest obstacle point n_q, the set of points in $\mathcal{E}_{free}$ within a distance of $d_P(q, n_q)$ from q is a disk when projected onto P. (b) The medial axis $MA(\mathcal{E})$ is drawn on the surfaces of $\mathcal{E}$.

5 THE MEDIAL AXIS IN MULTI-LAYERED ENVIRONMENTS

In this section, we first define the medial axis for walkable and multi-layered environments. Because our definitions do not require a particular subdivision into layers, they apply to both WEs and MLEs. Next, we show how to compute the medial axis in $O(n \log n \log k)$ time for an MLE with n boundary vertices and k connections.

5.1 Definition and Properties

To define the medial axis for walkable and multi-layered environments, we need a notion of distance and path length. We will use the direction-consistency of the WE to define *projected distances* in which height differences are ignored. Again, we acknowledge that this is not the same as the 3D distance on a surface. In Section 2.3, we have argued why this simplification is useful.

For two points s and g in a WE or MLE, let $\pi(s, g)$ be a path from s to g through $\mathcal{E}_{free}$ along the walkable surfaces. We define the *projected length* of $\pi(s, g)$ as the curve length of $\pi(s, g)$ when projected vertically onto the ground plane P. This projected path can intersect itself: For instance, consider a path along a spiral staircase.

Let $\pi^*(s, g)$ be a path from s to g with the smallest projected length (among all possible paths from s to g). We define the *projected distance* $d_P(s, g)$ between s and g as the projected length of $\pi^*(s, g)$. That is, $d_P(s, g)$ ignores any height differences along paths from s to g. Figure 4(a) shows an example of projected distances.

A shortest path $\pi^*(s, g)$ is *unobstructed* if it does not have any bending points around obstacles. The projection of an unobstructed path onto P is a single line segment, so its projected length is simply the 2D Euclidean distance between s and g (when projected onto P). The following properties hold:

Property 5.1 (Straight-line property). *The shortest path $\pi^*(q, n_q)$ from any point $q \in \mathcal{E}_{free}$ to any of its nearest boundary points n_q is unobstructed.*

Property 5.2 (Empty-circle property). *Let q be a point in $\mathcal{E}_{free}$ and let n_q be a nearest boundary point to q, at projected distance $d = d_P(q, n_q)$. For all points $q' \in \mathcal{E}_{free}$ for which $d_P(q, q') \leq d$, the shortest path $\pi^*(q, q')$ is unobstructed. When projected onto P, these points form a disk with radius d.*

If a WE has been converted to an MLE (i.e., if it has been subdivided into layers), then the nearest boundary point n_q to a point $q \in \mathcal{E}_{free}$ may lie on a different layer than q itself. Consequently, the empty disk around q may span multiple layers. This does not matter for our definitions.

We now define the medial axis based on the function d_P:

Definition 5.1 (Medial axis, multi-layered). For a walkable or multi-layered environment $\mathcal{E}$ with free space $\mathcal{E}_{free}$, let $ma(\mathcal{E})$ be the set of all points in $\mathcal{E}_{free}$ that have at least two equidistant nearest points on the boundary of $\mathcal{E}_{free}$ with respect to the projected distance function d_P. The medial axis $MA(\mathcal{E})$ is the topological closure of $ma(\mathcal{E})$.

Figure 4(b) shows the medial axis of an example WE. Because the remainder of this article is based on the projected distance function, we will often omit the term *projected* when discussing distances and path lengths.

If an environment $\mathcal{E}$ consists of a single layer L_i, then $MA(\mathcal{E}) = MA(L_i)$. Likewise, if we treat a 2D environment as an MLE with a single layer, then Definition 5.1 is actually a generalization of Definition 3.1, and we have obtained a single definition for the medial axis of 2D environments, MLEs, and WEs.

The medial axis becomes more interesting if $\mathcal{E}$ is a WE that overlaps itself when projected onto P or (equivalently) if $\mathcal{E}$ is an MLE that contains overlapping layers. In these cases, $MA(\mathcal{E})$ is typically not planar, but intuitively, it is *locally* similar to a 2D medial axis everywhere due to the straight-line and empty-circle properties. We will use these properties to prove that our construction algorithm for the multi-layered medial axis is correct.

5.2 Construction Algorithm Outline

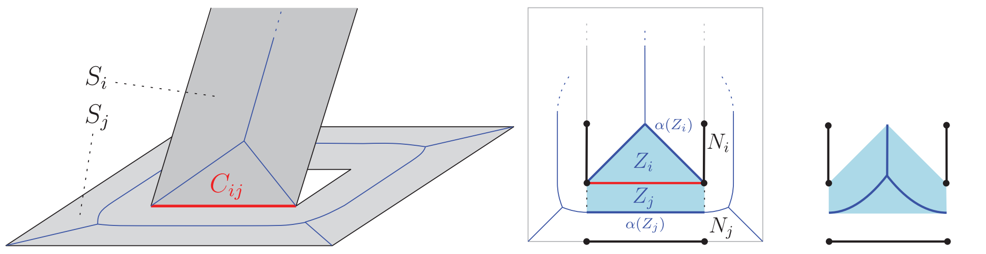
We now give an outline of our algorithm that computes the medial axis of an MLE $\mathcal{E}$. The result is also the medial axis of the corresponding WE. However, our algorithm makes use of the fact that the *2D* medial axis is easy to compute. For this reason, we assume that the environment has been partitioned into layers. We acknowledge that this is a necessary pre-processing step that can be solved using other algorithms [22]. Our construction algorithm consists of the following steps:

(1) For each individual layer L_i, project L_i onto P and compute its 2D medial axis, while treating all of its connections as *closed* impassable obstacles. This yields exactly the medial axis $MA(L_i)$ according to the projected distance function d_P, but under the assumption that each connection in C is an obstacle. The result for all layers combined is the medial axis of $\mathcal{E}$ under the same assumption. We denote this result by $MA(\mathcal{E}, C)$.

(2) Given $MA(\mathcal{E}, C')$ with $C' \subseteq C$, choose a closed connection $C_q \in C'$. By definition, the obstacle associated with C_q is a line segment when projected onto P. *Open* the connection C_q by removing its *interior* as an obstacle and repairing the medial axis in its neighborhood. (The endpoints of the connection will remain obstacles, because they are on the boundary of $\mathcal{E}_{free}$.) The result is the medial axis of $\mathcal{E}$ in which C_q is no longer treated as an impassable obstacle, i.e., it is $MA(\mathcal{E}, C'')$ with $C'' = C' \setminus \{C_q\}$. In $MA(\mathcal{E}, C'')$, there are new edges of the medial axis that pass through C_q, and the neighborhood of C_q is now connected. In Section 5.4, we will describe our algorithm for opening a connection.

(3) Repeat step 2 until all connections are open. The result is $MA(\mathcal{E}, \emptyset) = MA(\mathcal{E})$.

In short, we initially treat all connections as closed and then iteratively remove them as obstacles. Because connections project to line segments, opening a connection is essentially the deletion of a line segment Voronoi site [2] but with the extra difficulty that the neighborhood of the deleted site may span multiple layers. We will explain this further in Section 5.4. For now, it is sufficient to know that existing deletion algorithms for Voronoi sites in 2D [1, 10, 33] cannot immediately be applied. Section 5.4 will present an alternative algorithm.



(a) 3D view (b) 2D view (c) Medial axis of N_{ij}

Fig. 5. Opening a connection C_{ij} in an MLE. (a) Initially, C_{ij} is an obstacle on both sides, S_i and S_j. (b) 2D top view of the area around C_{ij}. The *influence zone* $Z_{ij} = Z_i \cup Z_j$ is shaded. The obstacle points $N_{ij} = N_i \cup N_j$ that are nearest to Z_{ij} are shown in bold black. (c) When opening C_{ij}, the medial axis changes only inside Z_{ij}. This medial axis M_Z is defined by N_{ij}.

5.3 Properties of a Closed Connection

To develop an algorithm for opening a closed connection, we must first study the properties of such a connection. Consider a closed connection between two layers L_i and L_j, as in Figure 5(a). We will now refer to this connection as C_{ij} to emphasize to which layers it is associated. This notation is not unique, because L_i and L_j may be connected via other connections as well. However, in our discussion of opening a single connection chosen by the main algorithm, it should be clear to which instance we are referring.

5.3.1 Sides. The connection is currently treated as an impassable obstacle between L_i and L_j. Thus, it occurs as an obstacle for the medial axis on two "sides." We define the *side* S_i as the set of all walkable surfaces and boundary points that are currently reachable from C_{ij} by starting in L_i. Likewise, the side S_j consists of all surfaces and obstacle points that can be reached from C_{ij} by starting in L_j. These sides are also annotated in Figure 5(a).

A side S_i at this point in our algorithm is not necessarily the same as a layer L_i in the environment. The side S_i includes at least the part of L_i that has C_{ij} on its boundary. If other connections are already open, then S_i may contain other layers as well. If sufficiently many connections have been opened such that L_i and L_j are already connected via another route, then S_i and S_j are even equal. However, for our algorithm, it does not matter which layers are already included in S_i or S_j, and it is useful to speak of two different sides of the connection.

5.3.2 Influence Zone. When we open C_{ij}, we effectively remove the *interior* of C_{ij}, denoted by *Int*(C_{ij}), as an obstacle from the environment. We do not remove the endpoints, because they will remain obstacles in the WE.

Therefore, we need to determine a new nearest obstacle for all points in the WE that were previously nearest to *Int*(C_{ij}). Let the *influence zone* Z_{ij} be the closure of the set of all points in $\mathcal{E}$ that currently have *Int*(C_{ij}) as a nearest obstacle. Observe from Figure 5(b) that Z_{ij} consists of two parts: one on side S_i and the other on side S_j. (Conceptually, it does not matter if S_i and S_j are already equal.) For convenience, we will refer to these parts as Z_i and Z_j, respectively.

Lemma 5.2. *If the interior of a connection C_{ij} is removed as an obstacle, then the medial axis changes only inside the influence zone Z_{ij}.*

Proof. By the definition of Z_{ij}, removing *Int*(C_{ij}) causes the nearest obstacle points to change only inside (and on the boundary of) Z_{ij}. After all, the other points in $\mathcal{E}_{free}$ were already closer to other obstacles. A consequence is that opening C_{ij} causes the *medial axis* to change only in Z_{ij}. □

Lemma 5.2 implies that $MA(\mathcal{E}, C')$ (the current medial axis with C_{ij} as an obstacle) and $MA(\mathcal{E}, C'')$ (the medial axis without C_{ij} as an obstacle, which we want to compute) are equal except in Z_{ij}. It is therefore useful to analyze the shape of Z_{ij}.

Lemma 5.3. *The influence zone Z_{ij} is bounded by the two lines perpendicular to C_{ij} through C_{ij}'s endpoints.*

Proof. (We will refer to these two lines as the *endpoint normals* of C_{ij}.) Consider any point $p \in \mathcal{E}_{free}$ that is *not* between or on the endpoint normals of C_{ij}. Such a point cannot be closest to $Int(C_{ij})$, because it must be closer to an endpoint of C_{ij} or to another obstacle in the environment. Therefore, p cannot be in Z_{ij}. □

Lemma 5.4. *Z_i and Z_j are both bounded by a sequence of medial axis arcs. Both sequences, denoted by $\alpha(Z_i)$ and $\alpha(Z_j)$, are uninterrupted and monotone with respect to the line supporting C_{ij}.*

Proof. We prove the lemma for Z_i; the proof for Z_j is analogous. Z_i is bounded by a set of medial axis arcs $\alpha(Z_i)$ that have a nearest obstacle point on C_{ij}. For every point z on $\alpha(Z_i)$, the nearest point c on C_{ij} can be reached via a line segment $\overline{zc}$ that is perpendicular to C_{ij}. If this were not true, then another obstacle would be in the way and c would not be a nearest obstacle point. Furthermore, c is a nearest obstacle point for all points on $\overline{zc}$, because this nearest obstacle cannot change when moving from z to c. Thus, $\overline{zc}$ lies entirely inside Z_i. Because Z_i consists of infinitely many line segments that all have an endpoint on C_{ij} and that are all perpendicular to C_{ij}, the boundary $\alpha(Z_i)$ is monotone with respect to C_{ij}.

Finally, the definition of an MLE enforces that $Int(C_{ij})$ does not intersect any obstacles. Because of this, every point on $Int(C_{ij})$ has some free space in its neighborhood: for each $c \in Int(C_{ij})$, the line segment $\overline{zc}$ exists and has non-zero length. The endpoints of C_{ij} are the only points where z and c can be equal. This proves that $\alpha(Z_i)$ is a single sequence of arcs. □

These lemmas have the following consequences.

Corollary 5.5. *The boundary of the influence zone Z_{ij} is a single closed loop consisting of $\alpha(Z_i)$, $\alpha(Z_j)$, and the endpoint normals of C_{ij}.*

Corollary 5.6. *The influence zone Z_{ij} can be projected onto the ground plane P without overlap. This projection is a single shape without holes (because it has only one boundary).*

5.3.3 Neighbor Set. Next, we determine which obstacles are required to update the medial axis inside Z_{ij}. On one side S_i of the connection, let N_i be the set of all obstacle points that are nearest to at least one point on $\alpha(Z_i)$, excluding $Int(C_{ij})$ itself. (On the other side S_j, let N_j be defined analogously.) These are the obstacle points that (together with C_{ij}) generate the arcs in $\alpha(Z_i)$. We exclude $Int(C_{ij})$ from N_i, because we will be removing this interior as an obstacle. We do explicitly include the *endpoints* of C_{ij} in N_i, because these will remain obstacles.

We define the *neighbor set* N_{ij} as the union of N_i and N_j. Informally, this set contains the "Voronoi neighbors" of the connection. N_{ij} consists of line segments and points on the boundary of $\mathcal{E}_{free}$. These are not necessarily the complete original boundary segments but only the parts that are actually relevant for Z_{ij}. Note that the neighbors can originate from many different layers and that they can even be (parts of) other connections that are still closed. A neighbor set is illustrated in Figure 5(b).

We will now prove that N_{ij} contains the obstacle points that define the medial axis in Z_{ij} when $Int(C_{ij})$ is removed. Lemma 5.7 proves that *all* points of N_{ij} are needed; Lemma 5.8 proves that *no other* points are needed.

Lemma 5.7. *When C_{ij} is opened, every obstacle point in N_{ij} is a nearest obstacle for at least one point in Z_{ij}.*

Proof. When the connection is still closed, every point $p \in N_{ij}$ is a nearest obstacle for at least one point z on the *boundary* of Z_{ij}, by definition. When C_{ij} is opened, p will still be nearest to z, because opening the connection only exposes z to obstacles that are farther away. Hence, there remains at least one point in Z_{ij} (namely z) for which p is a nearest obstacle. This means that all points of N_{ij} are required. □

Lemma 5.8. *When C_{ij} is opened, N_{ij} contains all possible nearest obstacle points for any point in Z_{ij}.*

Proof. We prove the lemma for Z_i; the proof for Z_j is analogous. For any point $p \in Z_i$, the nearest obstacle points currently lie in N_i or on C_{ij}, by definition. Removing the interior of C_{ij} cannot cause other obstacle points on the same side S_i to suddenly become nearest to p. The only remaining option is that an obstacle on the *other* side S_j becomes nearest to p. Such an obstacle must definitely lie in N_j: By definition, all other obstacle points of S_j were already not closest to C_{ij} itself, so they cannot be closest to a point *beyond* C_{ij}. Therefore, all possible nearest obstacle points for Z_i are included in N_i and N_j. □

5.4 Opening a Connection

To open a closed connection C_{ij}, we now know that we only need to update the medial axis inside the influence zone Z_{ij}. Thus, to convert the current medial axis $MA(\mathcal{E}, C')$ into $MA(\mathcal{E}, C'')$ with $C'' = C' \setminus \{C_{ij}\}$, it is sufficient to *only* compute $MA(\mathcal{E}, C'') \cap Z_{ij}$ and then replace $MA(\mathcal{E}, C') \cap Z_{ij}$ by it. We will refer to the new medial axis part, $MA(\mathcal{E}, C'') \cap Z_{ij}$, as M_Z for convenience.

Thus, our goal is to compute M_Z. Lemmas 5.7 and 5.8 guarantee that M_Z is defined by the obstacles in the neighbor set N_{ij}. An example of M_Z is shown in Figure 5(c).

Lemma 5.9. *The medial axis M_Z is a tree that can be projected onto P without overlap.*

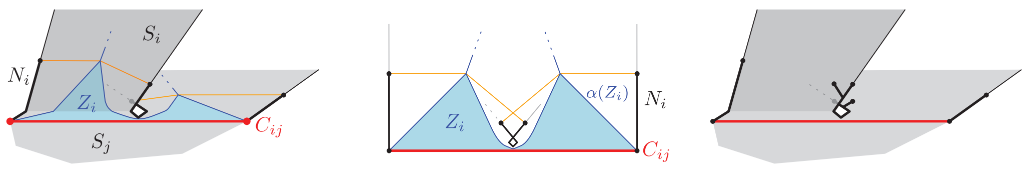
Proof. M_Z can only contain cycles if Z_{ij} contains holes; otherwise, there are no obstacles to circumnavigate. Furthermore, M_Z can only consist of multiple connected components if Z_{ij} consists of multiple disconnected shapes. Corollary 5.6 states that Z_{ij} is a single shape without holes. Therefore, M_Z is a single tree.

Corollary 5.6 also states that Z_{ij} is non-overlapping when projected onto P. Because M_Z lies entirely inside Z_{ij}, it can be projected onto P without overlap as well. □

In previous work [62], we computed M_Z by projecting all obstacles of N_{ij} onto the ground plane P and computing the 2D medial axis of this projection. This is equivalent to a deletion of a site from a 2D Voronoi diagram [10]; the algorithm takes $O(m \log m)$ time where m is the complexity of N_{ij}. Also, the algorithm is easy to implement by using any available library for Voronoi diagrams in 2D. We therefore still use it in our current implementation (Section 7).

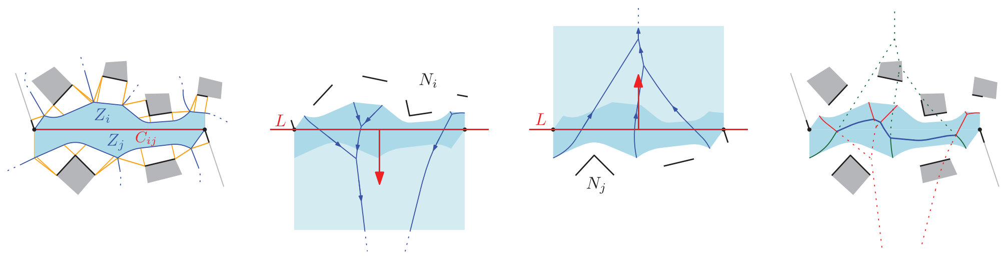
However, due to the multi-layered structure of $\mathcal{E}$, this algorithm does not work in all environments. A single common projection onto P may cause an obstacle of N_{ij} to influence parts of Z_{ij} to which it is actually not closest. Figure 6 shows an example in which the obstacles N_i on side S_i cannot be treated as a planar set.

We now propose an improved algorithm that uses projected distances *without* explicitly projecting all of N_i (or N_j) onto P at the same time. Our new approach starts at the boundary of Z_{ij} and traces the medial axis from there, based on the obstacles that are locally nearest. This avoids the problem of Figure 6. The new approach is outlined in Figure 7: Figure 7(a) shows the current situation with C_{ij} closed, and the other subfigures represent the algorithm for opening C_{ij}.



(a) 3D view (b) Projection onto P (c) Projection of N_i lifted to 3D

Fig. 6. Example in which projecting the entire neighbor set onto the ground plane P leads to problems. (a) One side S_i of a connection C_{ij} contains a ramp and a flat surface. (In this view, the flat surface is partly occluded by the ramp. The occluded boundary part is shown in dotted gray.) N_i (shown in bold) contains obstacle points from both parts. (b) A projection of the same situation onto P. (c) If we project all of N_i onto P at the same time, then we effectively treat the points of N_i as obstacles in all surfaces. This will yield an incorrect medial axis for Z_i.



(a) Before opening (b) $M_{Z,i}$ (c) $M_{Z,j}$ (d) Merging into M_Z

Fig. 7. We compute the medial axis M_Z in three steps. (a) The medial axis when C_{ij} is still closed. (b) $M_{Z,i}$ uses only the obstacles of N_i and assumes that Z_j extends to infinity. We compute it using a plane sweep on the ground plane P, starting with the sweep line L at C_{ij}, *without* explicitly projecting all of N_i onto P at the same time. (c) Analogously, $M_{Z,j}$ uses only N_j. (d) We merge the two parts to obtain M_Z.

Let $M_{Z,i}$ be M_Z under the assumption that there are no obstacles in N_j and that Z_j extends to infinity. Thus, $M_{Z,i}$ is defined solely by the obstacles of N_i. By the same arguments as before, the version of Z_{ij} in which Z_j extends to infinity is a simple shape when projected onto P (Corollary 5.6), and $M_{Z,i}$ is a tree that does not overlap in P either (Lemma 5.9). An example is shown in Figure 7(b).

Let $M_{Z,j}$ be defined analogously (Figure 7(c)). We compute $M_{Z,i}$ and $M_{Z,j}$ separately and then merge them to obtain M_Z (Figure 7(d)).

5.4.1 Computing a Single Part. To compute $M_{Z,i}$, we use the *plane sweep algorithm* by Fortune [11], which traces a Voronoi diagram (VD) by moving a horizontal sweep line L downwards. This algorithm is defined for sites in 2D, but we will show how to apply it to our multi-layered problem.

A thorough analysis of Fortune's algorithm has been given by de Berg et al. [3]. We will repeat the most important features. The algorithm maintains an x-monotone "beach line" consisting of bisector arcs; each arc is defined by the sweep line L and an input site above L. The endpoints of these beach line arcs (which are referred to as *break points*) are the centers of the largest empty disks in the environment that are tangent to L. The VD below the beach line is yet to be determined. As the sweep line moves downwards, the beach line changes, and its break points trace the edges of the VD. There are two types of events: *site* events when L reaches a new site, and *circle* events when L reaches the lowest point of a circle through three sites defining adjacent arcs on the beach line. Each event indicates that a site starts or stops generating a particular arc on the beach line.

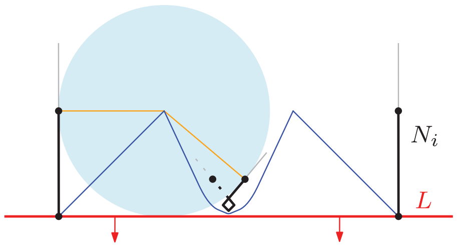


Fig. 8. The environment from Figure 6(b) when the sweep algorithm begins. A disk on the beach line may contain obstacles from other layers when projected onto P, but these are not *nearest* obstacles.

We apply Fortune's algorithm to our multi-layered problem by initializing the algorithm in such a way that all *site* events are already handled and all *circle* events can be processed just as in 2D. Assume without loss of generality that C_{ij} is horizontal and that Z_i lies above it. We start with a sweep line at the height of C_{ij} and initialize the beach line as the sequence of arcs $\alpha(Z_i)$. Lemma 5.4 states that this beach line is x-monotone, given that C_{ij} is horizontal. By the same argument, it will remain x-monotone during the sweep. After initialization, we move the sweep line downwards, and the algorithm proceeds exactly as if we were working in 2D.

The essential difference from a 2D problem is that the beach line now represents empty disks in the MLE and not on a single plane. To explain this further, Figure 8 shows the initial sweep line situation for the self-overlapping environment of Figure 6. An empty disk on the beach line is highlighted in blue. If we would project all of N_i onto P at the same time (as in our old algorithm [62]), then this disk would suddenly contain obstacles from other layers. However, these obstacles are *not nearest obstacles* according to our distance function d_P. This is why the old algorithm failed: It did not respect the empty-circle property of an MLE. The new algorithm succeeds because it *does* respect this property.

Each individual event in the sweep algorithm relies only on a point on the beach line and its nearest obstacles. Due to the straight-line and empty-circle properties given in Section 5.1, an event can be projected onto P, and its geometric computations will then work exactly as in 2D: Disks are still disks, and paths to nearest obstacles are still straight-line segments. The overall algorithm is a combinatorial sequence of events, so it does not require a projection of all events onto P at the same time. Therefore, the algorithm is not affected by the multi-layered structure of N_i.

We will now explain further which events occur and how they can be processed.

Site events. When the sweep begins, the endpoints of C_{ij} lie exactly on the sweep line. Both endpoints induce a site event that needs to be processed immediately. The following lemma implies that all other sites lie above C_{ij}, so there are no other site events.

Lemma 5.10. *Each point of N_i either lies above C_{ij} or is an endpoint of C_{ij}.*

Proof. We will prove that any obstacle point on side S_i that does *not* lie above C_{ij} cannot be in N_i. By definition, the interior of C_{ij} is *excluded* from N_i, and the endpoints of C_{ij} are *included.* Thus, we only need to consider the *other* obstacle points of S_i.

Recall that C_{ij} is still a closed obstacle when the set N_i is determined. This means that paths cannot yet go through C_{ij}. Let q be any obstacle point on side S_i that lies below or on the horizontal line C through C_{ij}. Let z be an arbitrary point in Z_i, and let c be the endpoint of C_{ij} that is nearest to z. Note that the shortest path $\pi^*(z, c)$ is unobstructed. We can prove that $\pi^*(z, q)$ is always longer:

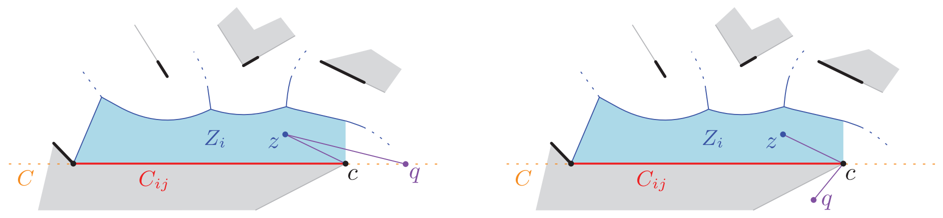


Fig. 9. One side S_i of a horizontal connection C_{ij}. An arbitrary point in $z \in Z_i$ is highlighted. Any obstacle point q below or on the supporting line C of C_{ij} cannot belong to the neighboring obstacles N_i. (a) The case in which $\pi^*(z, q)$ does not navigate around C_{ij}. (b) The case in which $\pi^*(z, q)$ navigates around C_{ij}. In both cases, an endpoint c of C_{ij} will be closer to z than q is. Therefore, q cannot be in N_i.

- If the line segment $\overline{zq}$ does not intersect C_{ij} when projected onto P, then the shortest path $\pi^*(z, q)$ is at best unobstructed, so it is at least as long as $\overline{zq}$. See Figure 9(a).
- Otherwise, $\pi^*(z, q)$ must navigate around C_{ij}, so it is at best a sequence of two line segments that bends around c (or the other endpoint of C_{ij}). See Figure 9(b).

In both cases, $\pi^*(z, q)$ is clearly longer than $\pi^*(z, c)$. Therefore, q cannot be nearest to any point in Z_i, and q cannot occur in N_i. □

Circle events. Initially, the potential circle events can be obtained by inspecting all 3-tuples of adjacent arcs in $\alpha(Z_i)$, just as in 2D. Because we look at adjacent arcs only, we will only trace Voronoi edges of obstacles that are actually Voronoi neighbors in the MLE. Since $M_{Z,i}$ will be a tree and Z_{ij} is planar when projected onto P, only adjacent Voronoi edges traced on the beach line can meet in a Voronoi vertex. Therefore, all circle events are discovered.

The effect of a circle event is the same as in 2D. Two of the empty disks on the beach line merge into one, a site locally disappears as a nearest site, and an arc disappears from the beach line. In terms of the VD, two Voronoi edges merge into a Voronoi vertex, and a new Voronoi edge starts being traced. The disappearance of an arc from the beach line induces two new 3-tuples of adjacent arcs. The circles tangent to the corresponding 3-tuples of sites may induce new circle events below the sweep line. These new events will be added to the event queue (sorted by y coordinate).

In summary, we generate the initial events using $\alpha(Z_i)$ as the beach line. After that, each individual event (i.e., each change of a nearest site) can be processed exactly as in 2D. Therefore, all events are recognized, and the algorithm correctly computes $M_{Z,i}$.

As shown in Figure 7(b), we end up tracing a tree of medial axis arcs, starting at the leaves and moving toward the root as we sweep downwards. By Lemma 5.10, the endpoints of C_{ij} are the lowest sites, so the final event occurs when the endpoints of C_{ij} become the only remaining nearest obstacles to the sweep line. These endpoints generate an infinite final edge that is perpendicular to C_{ij}.

5.4.2 Merging the Two Parts. We compute $M_{Z,j}$ similarly to $M_{Z,i}$ but by starting with $\alpha(Z_j)$ as the beach line and moving the sweep line upwards instead of downwards. Next, we merge $M_{Z,i}$ and $M_{Z,j}$ to obtain M_Z. We do this by using the merge procedure for Voronoi diagrams from Shamos and Hoey [58]. The merge procedure traverses the Voronoi cells of $M_{Z,i}$ and $M_{Z,j}$ simultaneously and builds a new monotone sequence of Voronoi edges between them. Afterwards, it removes the parts of $M_{Z,i}$ and $M_{Z,j}$ that are no longer needed. Figure 10 shows an adapted version of Figure 7(d) in which the result is easier to see.

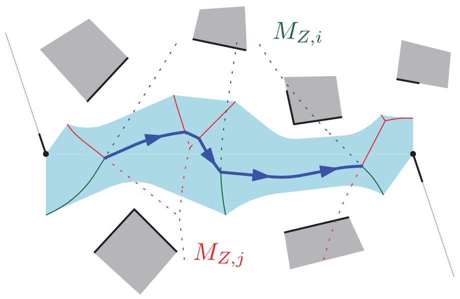


Fig. 10. Merging the medial axes $M_{Z,i}$ (green) and $M_{Z,j}$ (red) to obtain M_Z. The result is a combination of green edges, red edges, and a monotone sequence of newly generated edges (shown in bold blue) that can be computed from left to right.

The merge procedure is originally defined in 2D, but it relies on two main requirements that are also met in our multi-layered version of the problem. The first requirement is that $M_{Z,i}$ and $M_{Z,j}$ must be planar; we have shown in Section 5.4.1 that this requirement is fulfilled. The second requirement is that N_i and N_j must lie in separate half-planes. Lemma 5.10 implies that this holds: N_i and N_j are separated by C. The only exceptions are the endpoints of C_{ij} at which the merge starts and ends.

In each step of the merge procedure, there is one nearest site (a point or a line segment) $n_i \in N_i$ and one nearest site $n_j \in N_j$, and the new Voronoi edge is the bisector of n_i and n_j. This bisector arc ends when either of the nearest sites changes. Because the medial axes $M_{Z,i}$ and $M_{Z,j}$ are correct, they represent for all points in Z_{ij} the nearest sites from N_i and N_j, respectively. They therefore store all the information required to detect the changes in nearest sites. Furthermore, the computations in each step work exactly as in 2D due to the straight-line and empty-circle properties. Thus, the merge procedure [58] is not affected by the potential multi-layered nature of N_{ij}, and it correctly computes M_Z.

5.4.3 Summary. We now summarize our algorithm for opening a connection. Let $\mathcal{E}$ be an MLE, and let $MA(\mathcal{E}, C')$ be the medial axis computed so far, in which a non-empty set of connections $C' \subseteq C$ is closed. We open a connection $C_{ij} \in C'$ to obtain $MA(\mathcal{E}, C'')$, where $C'' = C' \setminus \{C_{ij}\}$. Opening C_{ij} works as follows:

(1) Remove the arcs from $MA(\mathcal{E}, C')$ that are nearest to the interior of C_{ij}. These are the arcs of $MA(\mathcal{E}, C')$ that bound the influence zone Z_{ij} described in Section 5.3.
(2) Compute $M_Z = MA(\mathcal{E}, C'') \cap Z_{ij}$ as described in Sections 5.4.1 and 5.4.2.
(3) Insert M_Z into $MA(\mathcal{E}, C')$ to obtain $MA(\mathcal{E}, C'')$.

5.5 Algorithm Correctness

The overall construction algorithm outlined in Section 5.2 first computes the medial axis with all connections as closed obstacles. It then iteratively opens a closed connection, using the algorithm from Section 5.4, until all connections are open. The following theorem states that this algorithm correctly computes the multi-layered medial axis.

Theorem 5.11. *Let $\mathcal{E}$ be an MLE. Computing $MA(\mathcal{E}, C)$ and then iteratively opening each connection in C as described in Section 5.4 yields the medial axis $MA(\mathcal{E})$.*

Proof. Each iteration of this algorithm starts with a correct medial axis $MA(\mathcal{E}, C')$ and computes a correct medial axis $MA(\mathcal{E}, C'')$ in which one more connection has been removed as an obstacle. By induction over the number of iterations, the final result is the correct medial axis $MA(\mathcal{E})$ in which all connections are traversable. □

The connections can be opened in any order without affecting the correctness of the algorithm. However, we will see in the next section that opening the connections in a particular order can affect the *running time* of the algorithm.

5.6 Algorithm Complexity

In this section, we analyze the asymptotic running time of our construction algorithm. The first step of the algorithm computes the medial axis of all layers with all connections closed. This can be achieved using a single 2D algorithm, because the medial axis consists of separate 2D components that do not yet influence each other. Lemma 4.3 has shown that the number of connections k is linear in the number of obstacle points n in the MLE. Thus, the presence of k connections does not affect the asymptotic complexity, and we essentially compute a 2D medial axis of an input with complexity $O(n)$. Section 3.3 has shown that this can be performed in $O(n \log n)$ time.

5.6.1 Running Time to Open One Connection.

Lemma 5.12. *A single connection C_{ij} can be opened in $O(m \log m)$ time, where m is the complexity of the neighbor set N_{ij}.*

Proof. Our algorithm for opening a connection starts with two instances of a sweep line algorithm [11]. Both instances take $O(m \log m)$ time, because they involve $O(m)$ circle events that need to be maintained in sorted order. Next, we perform one $O(m)$-time merge step of a divide-and-conquer algorithm [58]. Therefore, the total running time is $O(m \log m)$. □

In many practical scenarios, the connections and obstacles are spread throughout the environment, and m will be constant in most iterations of the algorithm. However, if many obstacles are close to the connection, m can be $\Theta(n)$.

5.6.2 Total Running Time. Based on Lemma 5.12, iteratively opening *all* connections takes $O(\sum_{i=0}^{k-1} m_i \log m_i)$ time in total, where m_i is the neighbor set complexity of the connection that is opened in iteration i. Note that the neighbor sets of closed connections can change during the algorithm, because obstacles may turn out to influence other connections when a nearby connection is opened. In other words, a neighbor set's complexity depends on what lies beyond the nearby connections that are already open. This suggests that the total construction time depends on the *order* in which the connections are opened.

In many environments, each obstacle will only affect the algorithm in a constant number of iterations regardless of this order, which means that the total construction time will remain $O(n \log n)$. However, there are environments in which opening the connections in an "unlucky" order leads to a worse construction time. The following lemma analyzes this.

Lemma 5.13. *There exists an environment with n obstacle vertices and k connections such that opening the connections in an inefficient order gives $\Theta(k)$ neighbor sets of complexity $\Theta(n)$.*

Proof. Figure 11 shows an example in which a ramp has been subdivided into a chain of small layers. All connections are close together, and the bottom connection has a row of $\Theta(n)$ neighboring obstacles on the ground plane. If the connections are opened from the bottom to the top, then the first neighbor set has complexity $\Theta(n)$. When the first connection is open, the same obstacles have become neighbors of the second connection, so the second neighbor set will also have complexity $\Theta(n)$. By repeating this argument, we see that this holds for each of the k iterations. □

When using the $O(m \log m)$-time algorithm from Section 5.4 in each iteration, the total running time for this unfortunate order becomes $O(kn \log n)$. In previous work [62], we reported this as

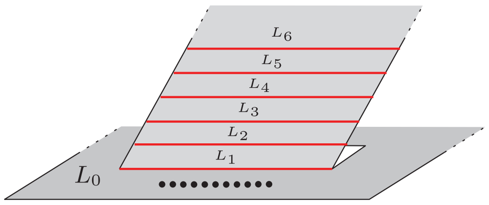


Fig. 11. A worst-case example for our incremental algorithm. A ramp has been subdivided into a sequence of small layers. The ground floor contains a row of $\Theta(n)$ obstacles. If we open the connections from bottom to top, then each connection will have $\Theta(n)$ obstacles in its neighbor set.

the worst-case running time of our algorithm. However, the following lemmas show that we can *always* construct the medial axis in $O(n \log n \log k)$ time by choosing an "easy" connection in each iteration. We begin by showing that the medial axis has linear size throughout the entire algorithm.

Lemma 5.14. *For any MLE, the medial axis has size $O(n)$ in each iteration of our algorithm.*

Proof. In the initial step of the algorithm, we compute a medial axis of $O(n)$ sites, which has size $O(n)$. Since opening a connection is analogous to deleting a Voronoi site, the asymptotic complexity of the graph cannot increase during the algorithm. □

Lemma 5.15. *When q connections are still closed, there is at least one connection with a neighbor set complexity of $O(\frac{n}{q})$.*

Proof. By Lemma 5.14, the medial axis always has $O(n)$ arcs. At any point in the incremental algorithm, every arc bounds the influence zone of at most two connections, namely one on each side of the arc. Therefore, the combined complexity of all influence zones (or, equivalently, of all neighbor sets) is $O(n)$. When this $O(n)$ complexity is shared by q connections, there must be at least one connection with a neighbor set complexity of $O(\frac{n}{q})$. □

Lemma 5.16. *The k connections can be opened in $O(n \log n \log k)$ total time.*

Proof. We will repeatedly open the connection with the smallest neighbor set complexity. To achieve this, we first compute the neighbor set complexities of all k closed connections. This can be done in $O(n)$ time by traversing the medial axis once and incrementing the complexity for a connection C_x whenever an arc has C_x as a nearest obstacle. Next, we sort the connections by complexity in a balanced binary search tree $\mathcal{T}$. This requires $O(k \log k)$ time, which is $O(n \log n)$, because k is $O(n)$.

Assume for now that we can maintain the sorting order in $\mathcal{T}$ such that we can always get the connection with the smallest complexity. Let C_q be this easiest connection when q connections are closed. By Lemma 5.15, it has a complexity of $O(\frac{n}{q})$. Using the algorithm of Section 5.4, we can open it in $O(\frac{n}{q} \log \frac{n}{q})$ time.

Next, we show that $\mathcal{T}$ can indeed be maintained efficiently. The neighbor sets of other closed connections can change due to opening C_q. However, any connection that is affected *must* be one of the neighboring obstacles of C_q. Therefore, the number of neighbor sets that can change is $O(\frac{n}{q})$. We can update the complexities of these neighbor sets in $O(\frac{n}{q})$ time: For each added or removed arc with a connection C_x as a nearest obstacle, we increment or decrement the neighbor set complexity for C_x. Afterwards, we update the search tree $\mathcal{T}$ by deleting and re-inserting all complexities that have changed. This takes $O(\frac{n}{q} \log q)$ time, because it requires $O(\frac{n}{q})$ update operations. Thus, opening C_q and updating $\mathcal{T}$ afterwards takes $O(\frac{n}{q}(\log \frac{n}{q} + \log q)) = O(\frac{n}{q} \log n)$ time.

Because $\mathcal{T}$ is maintained correctly, we can always open the connection with the lowest neighbor set complexity. For all k iterations combined, we obtain a running time of $O(\sum_{q=1}^{k} \frac{n}{q} \log n) = O(\sum_{q=1}^{k}(n \log n \cdot \frac{1}{q})) = O(n \log n \sum_{q=1}^{k} \frac{1}{q}) = O(n \log n \cdot H_k)$. Here, H_k is the kth harmonic number, which is known to be $\Theta(\log k)$. Therefore, the total running time to open all connections is $O(n \log n \log k)$. □

Combined with the $O(n \log n)$ running time for the first step (i.e., computing the medial axis of each layer with all connections closed), we obtain the following result.

Theorem 5.17. *The medial axis of an MLE with n obstacle vertices and k connections can be computed in $O(n \log n \log k)$ time.*

5.7 Storage Complexity

Finally, we give the storage complexity of the multi-layered medial axis when all connections have been opened. It follows immediately from Lemma 5.14: The complexity is $O(n)$ at each point in the algorithm, including at the end.

Theorem 5.18. *The medial axis of an MLE with n obstacle vertices and k connections has a storage complexity of $O(n)$.*

6 APPLICATION: THE EXPLICIT CORRIDOR MAP

The Explicit Corridor Map (ECM) is a navigation mesh that is closely related to the medial axis. In this section, we give an improved definition of the ECM, we analyze its complexity, and we show several useful geometric operations for path planning.

6.1 Definition

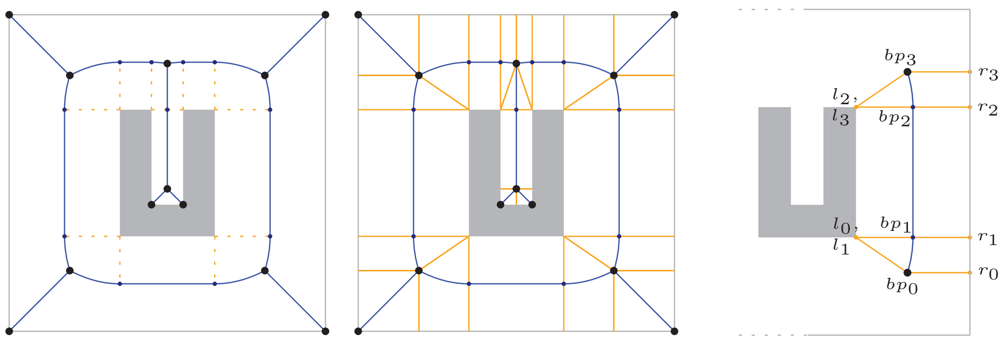
The ECM is a graph representation of the medial axis annotated with nearest-obstacle information. It describes each medial axis arc and its surrounding free space in an efficient manner. As such, it is a compact navigation mesh that can be used to find paths for characters of any radius.

Definition 6.1 (Explicit Corridor Map). For a 2D environment, WE, or MLE, the Explicit Corridor Map $ECM(\mathcal{E})$ is an extended representation of the medial axis $MA(\mathcal{E})$ as an undirected graph $G = (V, E)$ with the following properties:

- V is the set of true vertices of the medial axis (i.e., all medial axis vertices except those of degree 2).
- E is the set of edges of the medial axis.
- Each edge $e_{ij} \in E$ represents the medial axis arcs between two true vertices $v_i, v_j \in V$. It is represented by a sequence of $n' \geq 2$ bending points[1] $bp_0, \ldots, bp_{n'-1}$ where $bp_0 = v_i$, $bp_{n'-1} = v_j$, and $bp_1, \ldots, bp_{n'-2}$ are the remaining semi-vertices along the edge.
- Each bending point is a medial axis vertex annotated with nearest-obstacle information. A bending point bp_k on an edge stores its two nearest obstacle points l_k and r_k on the left and right side of the edge, respectively.

Figure 12(b) shows the ECM of our example environment. Since the ECM is an undirected graph, any edge e_{ij} could also be described as an edge e_{ji}, with the list of bending points reversed and all left and right obstacle points swapped. Furthermore, a true vertex occurs as the first or last bending point for each of its incident edges. Each such bending point has its own sense of left

[1]We originally referred to bending points as *event points* [14].



(a) Medial axis (b) Explicit Corridor Map (c) ECM edge

Fig. 12. (a) The medial axis of a 2D environment, repeated from Figure 2(b). (b) The ECM is a medial axis with nearest-obstacle annotations, shown as orange line segments between vertices and their nearest obstacle points. These segments are not edges in the graph. (c) Details of an ECM edge with four bending points. Each bending point bp_k stores its position p_k and its nearest obstacle points l_k and r_k.

and right; together, they store all nearest obstacle points for the true vertex. Thus, it is sufficient to store only two obstacle points for each bending point. We also emphasize that the orange line segments in Figure 12(b) are *not* graph edges; they merely denote the relation between bending points and their nearest obstacles. Figure 12(c) shows the details of an ECM edge.

Annotating the medial axis with nearest-obstacle information has many advantages. One advantage is that the *clearance* (the distance to the nearest obstacle) is known at each bending point. This enables path planning for characters of any radius; that is, we do not have to inflate the obstacles using Minkowski sums for a particular radius.

Our definition of the ECM applies not only to 2D environments but also to walkable and multi-layered environments without requiring any adjustments. In a WE or MLE, the nearest obstacle to a point $q \in \mathcal{E}_{free}$ may lie in another layer than q itself. Therefore, a nearest obstacle point to an ECM bending point bp_k may lie in another layer than bp_k. This does not change the ECM's definition or complexity. Due to the straight-line property, the path from a bending point to any of its nearest obstacle points is a line segment when projected onto P.

6.2 Complexity

We have shown that the medial axis has $O(n)$ complexity in a 2D environment with n obstacle vertices (Section 3.3), in a WE with n boundary vertices (Section 5.7), and in an MLE with n boundary vertices (regardless of the number of connections). The next lemma states that the medial axis can easily be converted to an ECM of the same complexity.

Lemma 6.2. *A medial axis with complexity $O(n)$ can be converted to an ECM of complexity $O(n)$ in $O(n)$ time.*

Proof. The ECM converts medial axis vertices to bending points by adding nearest-obstacle annotations. A degree-2 vertex becomes a single bending point, and any other vertex receives a separate bending point for each incident edge. The nearest-obstacle annotations can easily be added in a post-processing step (or even during the construction algorithm itself). After all, a medial axis arc is defined by an obstacle part (a line segment or a point) on both sides of the arc. The two nearest obstacle *points* for a bending point bp_k are simply the nearest points to bp_k on these obstacle parts. Thus, adding these annotations takes constant time per bending point.

What remains to be analyzed is the total number of bending points. Each medial axis vertex of degree 1 or 2 occurs as a bending point exactly once. A vertex of degree $d \geq 3$ occurs as a bending point d times: once for each edge that contains this vertex as an endpoint. Since the sum of all vertex degrees is $O(n)$, there are $O(n)$ bending points in total, each of which requires $O(1)$ storage. Thus, the ECM adds a linear amount of information to the medial axis in linear time. □

The remainder of this section defines a number of operations on the ECM that are useful for path planning. These concepts apply to both 2D and multi-layered environments. For more information on how these concepts fit into a generic crowd simulation framework, we refer the reader to an overview [65].

6.3 Computing Nearest Obstacles and Retractions

The ECM event points and their nearest-obstacle annotations subdivide the free space $\mathcal{E}_{free}$ into non-overlapping polygonal cells. This subdivision has $O(n)$ edges and vertices. We can therefore perform *point-location queries* to find out in which ECM cell a query point is located. Such queries can be answered in $O(\log n)$ time using, e.g., a trapezoidal map, which can be built in $O(n \log n)$ time [3]. For MLEs, we create a separate point-location data structure for each layer. A query point in an MLE is specified by a 2D point and a layer ID.

Once the cell containing a query point p has been determined, we can compute the *nearest obstacle point* $np(p)$ in constant time, because each cell has constant complexity. In a crowd simulation application, we can use this to easily determine how far each character is removed from the nearest boundary.

Points in the free space can be *retracted* onto the medial axis. In robot motion planning, the term *retraction* is used for a function that maps points in $\mathcal{E}_{free}$ onto the medial axis [71], as well as for complete planning methods based on this principle [46]. We use the following definition:

Definition 6.3 (Retraction). For any point p in the free space $\mathcal{E}_{free}$, the *retraction* $Retr(p)$ is a unique projection of p onto the medial axis:

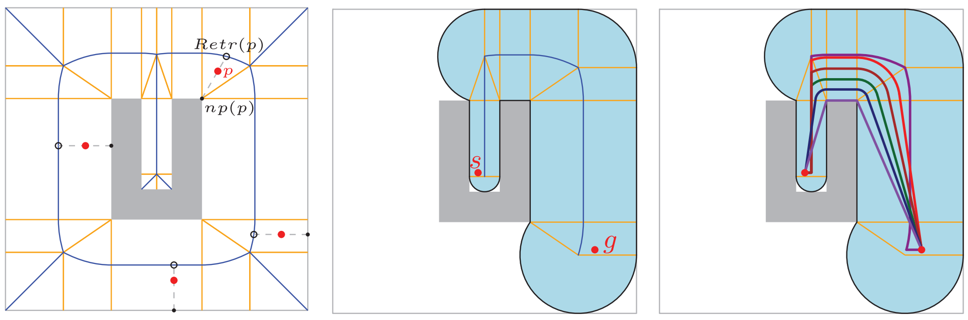
(1) If p lies on the medial axis, then $Retr(p) = p$.
(2) If p does not lie on the medial axis, then let l be the half-line that starts at $np(p)$ and passes through p. $Retr(p)$ is the first intersection of l with the medial axis.

Figure 13(a) shows examples of retractions. A retraction can be computed in $O(\log n)$ time by using a point-location query followed by constant-time geometric operations.

6.4 Computing a Path

To plan a path for a disk-shaped character in the ECM, we first find a path along the medial axis that has sufficient clearance. This is equivalent to the *retraction method* for motion planning [46]: Given a start position s and a goal position g in $\mathcal{E}_{free}$, we compute their retractions, and then we compute an optimal path on the medial axis from $Retr(s)$ to $Retr(g)$ using the A* search algorithm. This search is efficient, because the medial axis is a sparse graph compared to, e.g., a grid. The clearance information stored in each ECM bending point allows us to precompute the *minimum clearance* along each edge. The search can then skip edges for which the clearance is too low for our disk to pass through.

The free space around a medial axis path can be described using a *corridor*, which is the sequence of ECM cells along the path combined with the maximum-clearance disks at its ECM vertices [14]. Figure 13(b) shows an example.



(a) Retractions (b) ECM path + corridor (c) Indicative routes

Fig. 13. (a) Examples of query points (shown as large dots), their nearest obstacle points (small dots), and their retractions (circles). (b) Given two positions s and g, the retraction method is used to compute a path from s to g along the medial axis. A corridor describes the free space around this path. (c) Within the corridor, we can compute various types of indicative routes, e.g., with an amount of preferred clearance.

6.5 Computing an Indicative Route

An ECM path can be converted into an *indicative route*: a curve for the character to follow. Various types of indicative routes can be obtained in $O(m)$ time, where m is the number of ECM cells along the path. For instance, we can use a funnel algorithm to obtain the *shortest path* within a corridor, while keeping a preferred distance to obstacles whenever possible [14]. Examples are displayed in Figure 13(c). It is also easy to compute indicative routes that stay on the left or right side of the free space (or any interpolation of these extremes). Varying the "side preference" among characters is a convenient way to obtain diversity in the crowd.

6.6 Dynamic Updates

In dynamic environments, large obstacles can appear or disappear during the simulation. In previous work [63], we have presented algorithms that update the ECM locally due to the insertion or deletion of a convex polygonal obstacle P. A dynamic insertion takes $O(m + \log n)$ time, where m is the combined complexity of the neighboring obstacles for P. A dynamic deletion requires $O(m \log m + \log n)$ time. For more details, we refer interested readers to the corresponding publication [63]. After a dynamic event, a character can efficiently re-plan a new optimal path in the updated mesh based on its previous path [64].

7 IMPLEMENTATION

We have implemented the Explicit Corridor Map as part of a crowd simulation framework [65]. The software was written in C++ in Visual Studio 2013.

To compute the medial axis and ECM, we have integrated two different libraries for computing Voronoi diagrams: Vroni [19] and a package of Boost [7]. Since the Boost Voronoi library requires integer coordinates as input, we multiply all coordinates by 10,000 and round them to the nearest integer. For convenience, we use these rounded coordinates in Vroni as well. We use meters as units, so this scaling implies that we represent all coordinates within a precision of 0.1mm.

In an earlier publication [62], we used an approximating GPU-based ECM implementation based on the work of Hoff et al. [24], However, we will not report the details of this approach, because it has proven to be less efficient and less practical than the other implementations.

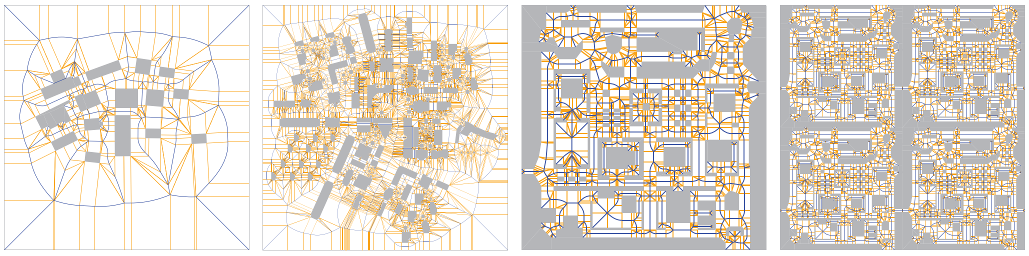


Fig. 14. The set of 2D environments and their ECMs. *Zelda4x4* and *Zelda8x8* are not shown, because they are very large and they are structurally similar to *Zelda2x2*.

Both Vroni and Boost assume that the input sites are interior-disjoint line segments. In practice, environments are often drawn by hand and may contain overlapping geometry. Therefore, before computing the ECM, we use another component of Boost to convert obstacles to interior-disjoint segments, using the scaled integer coordinates described earlier. After computing the ECM, we remove all graph components that lie inside obstacle polygons. These steps will be included in our time measurements.

In some environments, the medial axis may contain edges that run across many layers. We ensure that each edge can be associated with a single layer, mainly for visualization purposes. We do this by splitting each edge wherever it intersects one of the (now opened) connections. Computing these intersections takes extra time, and it can increase the worst-case complexity of the graph to $O(kn)$ if many edges intersect many connections, such as in Figure 11. We consider this post-processing to be optional and not part of the main algorithm.

Section 8.2 will show that our implementation of the multi-layered ECM construction algorithm is very fast in practice. For future work, there are two potential improvements. First, we currently open the connections in the order in which they are listed in the environment, which is not necessarily optimal. Second, we open the connections using our old algorithm [62], so we cannot yet handle self-overlap near connections such as in Figure 6. However, these theoretical issues are not a problem for any of the real-world environments in our test set.

For simplicity, we have implemented indicative routes as piecewise linear curves. Circular or parabolic arcs in an indicative route are approximated by sequences of line segments.

8 EXPERIMENTS

This section assesses the performance of our ECM implementations in a range of 2D and multi-layered environments. All experiments were run on a Windows 7 PC with a 3.20GHz Intel i7-3930K CPU, an NVIDIA GeForce GTX 680 GPU, and 16GB of RAM. Only one CPU core was used, except at the end of Section 8.2 where we will use multi-threading to improve the performance in MLEs.

8.1 Environments

The 2D environments are shown in Figure 14; more details can be found in the upper part of Table 1. *Military* is a simple environment with a small number of obstacles. *City* is a more complex virtual city. *Zelda* is an environment from a computer game. *Zelda2x2*, *Zelda4x4*, and *Zelda8x8* are adapted versions of *Zelda* that have been duplicated in a 2×2, 4×4, and 8×8 grid pattern. We have also used these environments in previous publications [63, 66].

The MLEs are shown in Figures 15–17 and are described in the lower part of Table 1:

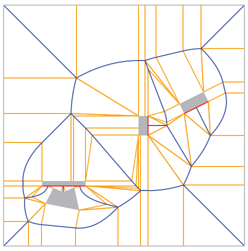
Table 1. Details of the Environments Used in Our Experiments

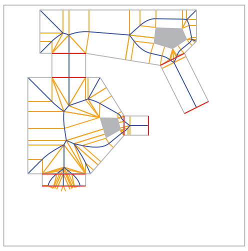
	Geometry		**Multi-layered**	
Environment	#Obstacle vertices	Size (m)	#Layers	#Connections
Military	108	200×200	1	0
City	2,102	500×500	1	0
Zelda	564	100×100	1	0
Zelda2x2	2,304	200×200	1	0
Zelda4x4	9,180	400×400	1	0
Zelda8x8	36,684	800×800	1	0
Ramps	147	100×100	7	8
Ramps2	422	100×100	7	8
Library	717	60×24	9	8
Station	2,242	153×111	34	64
Tower	6,058	35×35	17	30
Stadium	12,915	280×184	18	82
BigCity	49,476	500×500	113	196
BigCity2x2	197,884	1000×1000	449	784

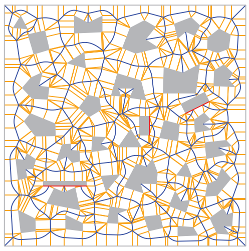
Note: The *Geometry* columns show the number of obstacle vertices and the physical width and height of the environment (in meters). The *Multi-layered* columns show the number of layers and connections of each environment.

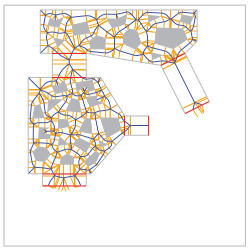
- *Ramps* consists of three flat layers connected by four ramps. Each ramp is modeled as a separate layer for simplicity. Figure 1 at the beginning of this article shows *Ramps* and its ECM in 3D.
- *Ramps2* is a version of *Ramps* in which we have added 56 polygonal obstacles to the flat layers.
- *Library* is a simplified model of the Utrecht University library.
- *Station* is a model of a train station with one main hall and one layer containing all platforms; these two layers are connected by 32 ramps. Again, each ramp has been modeled as a separate layer.
- *Tower* is a complex multi-storey apartment building.
- *Stadium* is a model of an American football stadium with many staircases and obstacles. Since it has been drawn manually based on real-world data, it contains small gaps that generate disconnected graph components. It also features sequences of nearly-collinear points that generate medial axis edges when the input coordinates have been rounded and scaled. These graph elements seem redundant, but they are correct in our scaled integer coordinate system.
- *BigCity* is a combination of the 2D city environment, six instances of *Tower*, and two instances of *Library*. The towers are highly detailed compared to the rest of the environment. Voxel-based navigation mesh algorithms would require a very high resolution to capture all details.
- *BigCity2x2* consists of four tiled instances of *BigCity*. It measures 1km^2 and contains 784 connections.

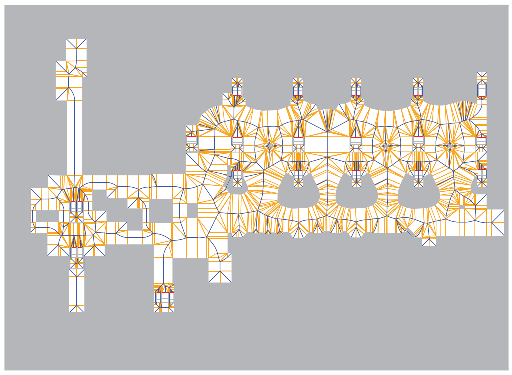
These environments were chosen to broadly reflect a range of complexities in terms of the number of layers, obstacle vertices, and connections. In previous work [62], we also explored theoretically challenging "toy examples," such as copies of an environment with an increasing number of


(a) Ramps


(b) Ramps (continued)


(c) Ramps2


(d) Ramps2 (continued)


(e) Station

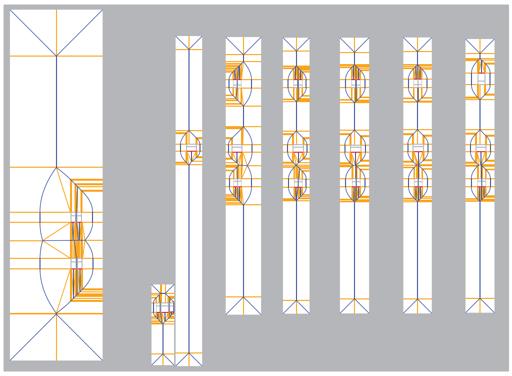

(f) Station (continued)

Fig. 15. 2D views of some of the MLEs used in our experiments. For *Station*, all stairways and escalators were modeled as separate layers; these layers are not shown.

connections. In this article, we choose to focus on realistic environments instead. For more environments, including ones that have been used in research on other navigation meshes, we refer the reader to our recent comparative study [66].

8.2 Computing the ECM

For the *2D environments*, the ECM complexities and construction times are shown in the upper part of Table 2. Vroni and Boost handle degenerate cases such as degree-4 vertices differently, which leads to slightly different ECM complexities for both implementations. The complexities in Table 2 were taken from the Boost version.

The Vroni-based implementation was faster than the Boost-based implementation in all environments. For the most complex 2D environment, *Zelda8x8*, computing the ECM took just under 1s when using Vroni. Hence, even complex ECMs can be computed quickly. This allows the navigation mesh to be generated interactively (e.g., when loading a game level or when used in a tool for designing environments).

For the *MLEs*, the ECM complexities and construction times are shown in the lower part of Table 2. While it can be seen that a multi-layered ECM takes more time to compute than a 2D ECM of the same complexity, the construction time is still well under a second for all environments except the two *BigCity* variants. For the largest environment, *BigCity2x2*, the construction takes about 9s when using Vroni.

The Boost implementation for Voronoi diagrams is thread-safe, so we can use *multi-threading* to compute the initial ECMs of all layers in parallel. The running times for this multi-threaded

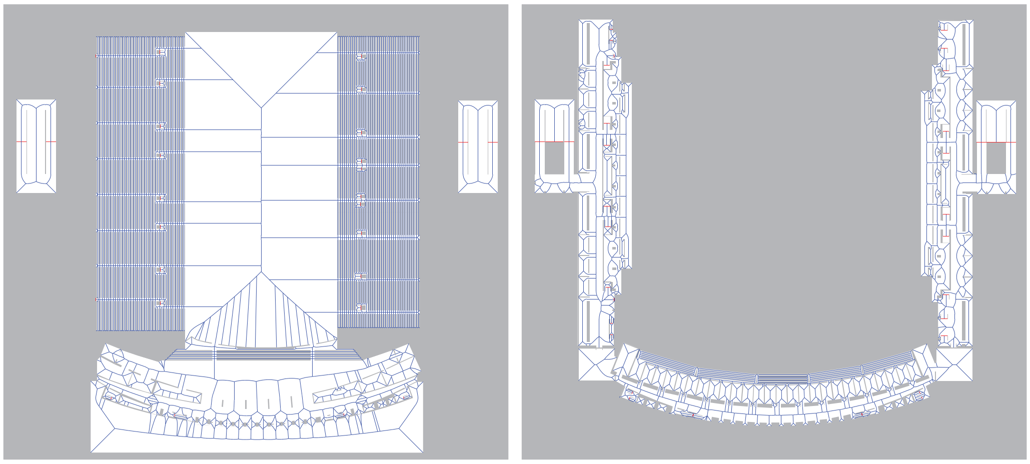
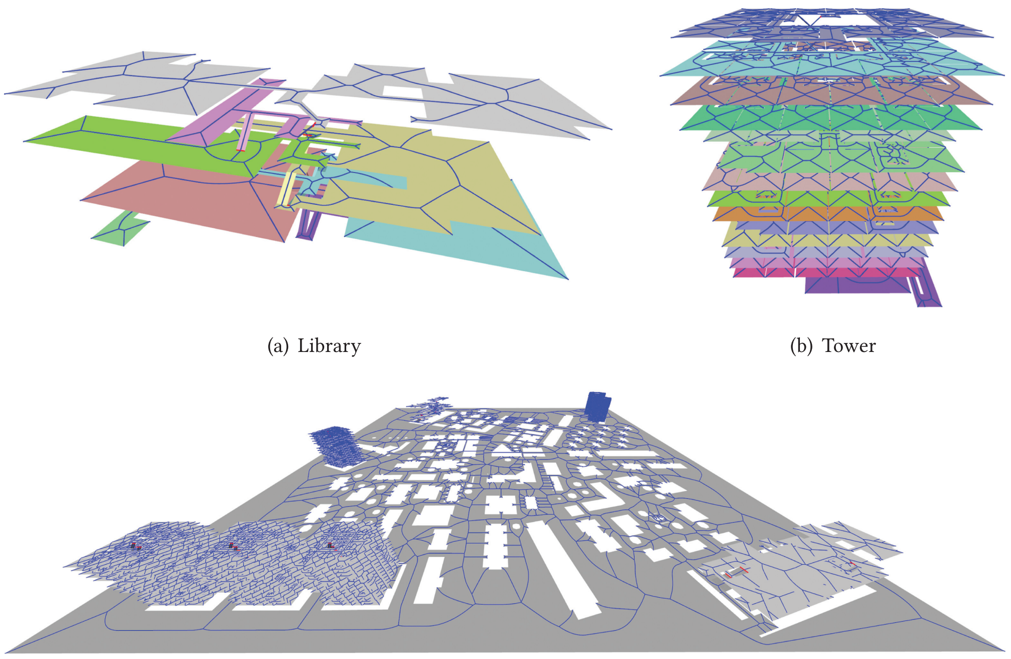

(a) Stadium (b) Stadium (continued)

Fig. 16. 2D views of several layers of the *Stadium* environment. Nearest-obstacle annotations of the ECM have been omitted for clarity. Some inaccuracies in the input geometry are too small to see in this image.


(a) Library (b) Tower

(c) BigCity

Fig. 17. 3D views of the *Library*, *Tower*, and *BigCity* environments and their medial axes. For clarity, the nearest-obstacle annotations of the ECM have been omitted. We have now colored the free space rather than the obstacle space. The gray obstacles from the 2D *City* environment are now modeled as holes in the free space of *BigCity*. *BigCity2x2* is not shown, because it is very large and structurally similar to *BigCity*.

Table 2. Details of the ECMs for Our Experiments

	ECM complexity			ECM time (ms)		
Environment	#Vertices	#Edges	#BPs	Vroni	Boost	Boost (MT)
Military	56	71	288	3.5 [0.1]	6.7 [0.1]	–
City	1,442	1,621	6,306	70.9 [0.3]	130.0 [0.7]	–
Zelda	296	351	1,258	14.9 [0.1]	23.0 [0.2]	–
Zelda2x2	1,184	1,408	5,082	59.2 [0.5]	92.0 [0.6]	–
Zelda4x4	4,720	5,624	20,329	235.4 [2.4]	367.7 [1.6]	–
Zelda8x8	18,848	22,480	81,365	997.4 [5.4]	1529.1 [3.8]	–
Ramps	54	61	181	5.8 [0.2]	9.4 [0.3]	7.6 [0.1]
Ramps2	228	290	1,118	16.3 [0.4]	29.5 [0.6]	21.2 [0.1]
Library	219	222	599	14.7 [0.3]	20.2 [0.4]	9.2 [0.1]
Station	660	768	2,804	68.6 [0.3]	97.3 [0.5]	74.3 [0.3]
Tower	4,948	4,979	14,407	248.8 [2.2]	383.4 [1.3]	110.1 [3.4]
Stadium	6,303	7,754	26,323	442.4 [8.2]	572.2 [1.6]	263.5 [4.7]
BigCity	32,264	32,652	104,002	2168.6 [19.6]	3430.0 [10.9]	925.7 [29.6]
BigCity2x2	129,147	130,702	416,411	8972.5 [32.7]	14287.0 [41.7]	4219.3 [91.9]

Note: The *ECM complexity* columns show the number of vertices, edges, and bending points in the ECM computed using Boost. The *ECM time* columns show the ECM construction time for the three implementations: Vroni, Boost, and Boost with five parallel threads (for MLEs). All times are in milliseconds and have been averaged over 10 runs. Standard deviations are shown between square brackets.

Boost version are also shown in Table 2. We used OpenMP with five parallel threads and dynamic scheduling. This version performed particularly well in environments with many complex layers; in particular, the ECM of *BigCity2x2* was computed in approximately 4.2s. Standard deviations among running times were higher, because the threads were scheduled in an unpredictable way. Still, this implementation shows that multi-threading is a promising addition.

8.3 Path Planning

In each environment, we have computed indicative routes between 10,000 pairs of random start and goal points. For each point, we first chose a random layer (if applicable) such that the probability of a layer being chosen was proportional to its surface area. The point itself was then chosen by uniformly sampling in the layer's bounding box until an obstacle-free point was found.

For each query pair (s, g), we computed the shortest path between $Retr(s)$ and $Retr(g)$ on the medial axis using A* search, with the 2D Euclidean distance to $Retr(g)$ as a heuristic. We then converted this path to a short indicative route with a preferred distance of 0.5m to obstacles.

Table 3 show the average running times of the complete path planning query per environment. The running time depends heavily on the complexity of the resulting path; this explains the high standard deviations. It can be seen that queries require only a few milliseconds on average in the most complex environments. Thus, the ECM allows real-time path planning for large crowds of characters with individual goals. In very complex environments, grid-based planning would be much slower, because a high grid resolution would be required to capture all details.

We refer the reader to previous work [65] for experiments on crowd simulation. This previous publication has shown that the running time of a simulation *without* collision avoidance scales linearly with the number of characters. Collision avoidance is inherently the most expensive step, because characters need to find their neighbors and respond to their movement. By using four CPU cores and eight parallel threads, we can currently simulate around 15,000 characters in real

Table 3. Results of the Path Planning Experiments

Environment	Planning time (ms)
Military	0.21 [0.15]
City	1.12 [0.70]
Zelda	0.41 [0.21]
Zelda2x2	0.96 [0.47]
Zelda4x4	1.99 [1.01]
Zelda8x8	4.65 [2.75]
Ramps	0.13 [0.08]
Ramps2	0.37 [0.18]
Library	0.53 [0.32]
Station	0.79 [0.48]
Tower	1.58 [0.68]
Stadium	2.22 [1.43]
BigCity	3.03 [2.55]
BigCity2x2	8.44 [7.59]

Note: The *Planning time* column shows the running time to compute a path, averaged over 10,000 random queries. All times are in milliseconds. Standard deviations are shown between square brackets.

time with collision avoidance. The ECM framework has a small memory footprint that allows simulations of at least 1 million characters. It has been successfully used to simulate crowded real-life events such as the 2015 *Tour de France* opening in Utrecht (The Netherlands) and evacuations of future metro stations in Amsterdam.

9 CONCLUSIONS AND FUTURE WORK

In robotics, simulations, and gaming applications, virtual walking characters often need to compute and traverse paths through an environment. A navigation mesh is a subdivision of the free space into polygons that allows real-time path planning for crowds of characters.

With this application area in mind, we have studied the medial axis for 3D environments with a consistent direction of gravity. A walkable environment (WE) describes where characters can walk. A multi-layered environment (MLE) is a WE that has been subdivided into 2D layers connected by k connections with certain geometric properties. We have defined the medial axis for WEs and MLEs based on projected distances on the ground plane. We have presented an algorithm that computes this medial axis by initially treating all connections as closed obstacles and then opening them incrementally. Compared to previous work [62], we have presented a new and correct algorithm for opening a connection, and we have improved the overall running time to $O(n \log n \log k)$ by opening the connections in an efficient order.

We have presented an improved definition of the Explicit Corridor Map (ECM), which is a navigation mesh based on the medial axis. The ECM enables path planning for disk-shaped characters of any radius. It supports efficient geometric operations such as retractions, nearest-obstacle queries, dynamic updates, and the computation of short paths with preferred clearance to obstacles.

Our implementation computes the ECM efficiently for large 2D and multi-layered environments. The ECM supports important applications and operations, such as path planning and dynamic updates. It is a useful basis for simulating large crowds of heterogeneous characters in real time [65].

We have given an $O(m \log m)$-time algorithm for opening a connection bounded by m medial axis arcs. This algorithm is not necessarily optimal. For 2D Voronoi diagrams, a line segment site can be removed in $O(m)$ time [33]. While this linear-time algorithm cannot immediately be applied to our multi-layered domain, it suggests that a $O(m)$-time solution for opening a connection might exist. Finding such an algorithm is a challenge for future work. It would allow us to improve the overall construction time for the multi-layered ECM to an optimal $O(n \log n)$.

A drawback of navigation meshes in general is that the shortest path in the dual graph (or, in our case, the medial axis) is not necessarily homotopic to the shortest path in the entire environment. Therefore, even a shortened path as described in Section 6.5 can be longer than the overall shortest path. For future work, we want to analyze path lengths in the ECM and investigate how to improve them. One option is to combine the ECM with a visibility graph, which yields shortest paths but has a size of $O(n^2)$ [15, 69].

While it is common to use projected distances for navigation meshes, we would like to investigate how navigation meshes can be extended to encode information about *height differences* as well. As explained in Section 2.3, one could simply use the WE itself as a data structure for path planning, but this approach lacks some of the advantages of the ECM (such as the ability to compute paths for disks of any radius).

Currently, state-of-the art 3D navigation meshes convert a raw 3D environment to a WE by using voxel-based filtering algorithms. Our comparative study of navigation meshes [66] has shown that this approach is not ideal: It requires parameter tuning, it is not scalable to large environments, and it sometimes sacrifices precision. We are therefore interested in developing *exact* algorithms for obtaining walkable and multi-layered environments from arbitrary 3D geometry. Note that an exact algorithm will recognize very small details in the environment that may not be relevant for the application, especially when using imprecise real-world data.

If a WE is given as a set of triangles, then converting it to an MLE with a minimum number of connections is NP-hard in the number of triangles [21]. However, there may be other criteria by which an MLE can be considered “optimal,” such as the total length of all connections combined. Finding an optimal MLE (or a constant-factor approximation of it) based on such criteria is an interesting topic for future work.

We are also interested in lifting other 2D data structures to the multi-layered domain, including visibility graphs and related concepts [52, 69]. We believe that MLEs give rise to an interesting new class of problems for future research.

ACKNOWLEDGMENTS

We thank Elena Khramtcova for fruitful discussions on deletions in Voronoi diagrams, Arne Hillebrand for helping us construct the 3D environments, and Incontrol Simulation Solutions for supplying the *Stadium* environment.

REFERENCES

[1] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor. 1989. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discr. Comput. Geom.* 4 (1989), 591–604.

[2] F. Aurenhammer, R. Klein, and D. T. Lee. 2013. *Voronoi Diagrams and Delaunay Triangulations.* World Scientific.

[3] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer.

[4] J. P. van den Berg, S. J. Guy, M. C. Lin, and D. Manocha. 2011. Reciprocal n-body collision avoidance. In *Proceedings of the 14th International Symposium on Robotics Research.* 3–19.

[5] G. Berseth, M. Kapadia, and P. Faloutsos. 2015. ACCLMesh: Curvature-based navigation mesh generation. In *Proceedings of the 8th ACM SIGGRAPH International Conference on Motion in Games.* 97–102.

[6] H. Blum. 1967. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, W. Whathen-Dunn (Ed.). MIT Press, Cambridge, MA, 362–380.

[7] Boost. 2018. The Boost C++ library. Retrieved from http://www.boost.org/.

[8] F. Chin, J. Snoeyink, and C. A. Wang. 1999. Finding the medial axis of a simple polygon in linear time. *Discr. Comput. Geom.* 21, 3 (1999), 405–420.

[9] L. Deusdado, A. R. Fernandes, and O. Belo. 2008. Path planning for complex 3D multilevel environments. In *Proceedings of the 24th Spring Conference on Computer Graphics.* 187–194.

[10] O. Devillers. 1999. On deletion in Delaunay triangulations. In *Proceedings of the 15th Annual ACM Symposium on Computational Geometry.* 181–188.

[11] S. Fortune. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica* 2 (1987), 153–174.

[12] M. Garber and M. C. Lin. 2004. Constraint-based motion planning using Voronoi diagrams. In *Algorithmic Foundations of Robotics V.* Springer, 541–558.

[13] F. M. García, M. Kapadia, and N. M. Badler. 2014. GPU-based dynamic search on adaptive resolution grids. In *Proceedings of the 31st IEEE International Conference on Robotics and Automation.* 1631–1638.

[14] R. Geraerts. 2010. Planning short paths with clearance using explicit corridors. In *Proceedings of the 27th IEEE International Conference on Robotics and Automation.* 1997–2004.

[15] S. K. Ghosh. 2007. *Visibility Algorithms in the Plane.* Cambridge University Press.

[16] P. J. Green and R. Sibson. 1978. Computing Dirichlet tessellations in the plane. *Comput. J.* 21, 2 (1978), 168–173.

[17] P. Hart, N. Nilsson, and B. Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybernet.* 4, 2 (1968), 100–107.

[18] D. Helbing and P. Molnár. 1995. Social force model for pedestrian dynamics. *Phys. Rev. E* 51, 5 (1995), 4282–4286.

[19] M. Held. 2011. VRONI and ArcVRONI: Software for and applications of Voronoi diagrams in science and engineering. In *Proceedings of the 8th International Symposium on Voronoi Diagrams in Science and Engineering.* 3–12.

[20] J. Hershberger and S. Suri. 1999. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.* 28, 6 (1999), 2215–2256.

[21] A. Hillebrand, J. M. van den Akker, R. Geraerts, and J. A. Hoogeveen. 2016. Performing multicut on walkable environments. In *Proceedings of the 10th International Conference on Combinatorial Optimization and Applications.* 311–325.

[22] A. Hillebrand, J. M. van den Akker, R. Geraerts, and J. A. Hoogeveen. 2016. Separating a walkable environment into layers. In *Proceedings of the 9th ACM SIGGRAPH International Conference on Motion in Games.* 101–106.

[23] M. Höcker, V. Berkhahn, A. Kneidl, A. Borrmann, and W. Klein. 2010. Graph-based approaches for simulating pedestrian dynamics in building models. In *eWork and eBusiness in Architecture, Engineering and Construction.* CRC Press, Boca Raton, FL, 389–394.

[24] K. E. Hoff III, T. Culver, J. Keyser, M. Lin, and D. Manocha. 1999. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques.* 277–286.

[25] C. Holleman and L. E. Kavraki. 2000. A framework for using the workspace medial axis in PRM planners. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 2. 1408–1413.

[26] N. S. Jaklin, A. F. Cook IV, and R. Geraerts. 2013. Real-time path planning in heterogeneous environments. *Comput. Anim. Virtual Worlds* 24, 3 (2013), 285–295.

[27] M. Kallmann. 2014. Dynamic and robust Local Clearance Triangulations. *ACM Trans. Graph.* 33, 5, Article 161.

[28] M. Kallmann and M. Kapadia. 2016. *Geometric and Discrete Path Planning for Interactive Virtual Worlds.* Morgan & Claypool.

[29] I. Karamouzas and M. H. Overmars. 2010. A velocity-based approach for simulating human collision avoidance. In *Proceedings of the 10th International Conference on Intelligent Virtual Agents.* 180–186.

[30] M. I. Karavelas. 2004. A robust and efficient implementation for the segment Voronoi diagram. In *Proceedings of the 1st International Symposium on Voronoi Diagrams in Science and Engineering.* 51–62.

[31] M. Kaul, R. C.-W. Wong, B. Yang, and C. S. Jensen. 2013. Finding shortest paths on terrains by killing two birds with one stone. *Proc. VLDB Endow.* 7, 1 (2013), 73–84.

[32] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* 12, 4 (1996), 566–580.

[33] E. Khramtcova and E. Papadopoulou. 2015. Linear-time algorithms for the farthest-segment Voronoi diagram and related tree structures. In *Proceedings of the 26th International Symposium on Algorithms and Computation.* 404–414.

[34] D. G. Kirkpatrick. 1979. Efficient computation of continuous skeletons. In *Proceedings of the IEEE 54th Annual Symposium on Foundations of Computer Science*. 18–27.

[35] S. Koenig and M. Likhachev. 2002. D* Lite. In *Proceedings of the 18th National Conference of Artificial Intelligence*. 476–483.

[36] J. J. Kuffner and S. M. LaValle. 2000. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings of the 17th IEEE International Conference on Robotics and Automation*. 995–1001.

[37] J. C. Latombe. 1991. *Robot Motion Planning*. Kluwer Academic.

[38] S. M. LaValle. 2006. *Planning Algorithms*. Cambridge University Press.

[39] D. T. Lee. 1982. Medial axis transformation of a planar shape. *IEEE Trans. Pattern Anal. Mach. Intell.* 4, 4 (1982), 363–369.

[40] D. T. Lee and R. L. Drysdale III. 1981. Generalization of Voronoi diagrams in the plane. *SIAM J. Comput.* 10, 1 (1981), 73–87.

[41] W. Lee and R. Lawrence. 2013. Fast grid-based path finding for video games. In *Advances in Artificial Intelligence*. Lecture Notes in Computer Science, Vol. 7884. Springer, 100–111.

[42] J.-M. Lien, S. L. Thomas, and N. M. Amato. 2003. A general framework for sampling on the medial axis of the free space. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 3. 4439–4444.

[43] M. Mononen. 2017. Recast Navigation. Retrieved from https://github.com/memononen/recastnavigation/.

[44] M. Moussaïd, D. Helbing, and G. Theraulaz. 2011. How simple rules determine pedestrian behavior and crowd disasters. *Proc. Natl. Acad. Sci. U.S.A.* 108, 17 (2011), 6884–6888.

[45] R. Narain, A. Golas, S. Curtis, and M. C. Lin. 2009. Aggregate dynamics for dense crowd simulation. *ACM Trans. Graph.* 28, 5 (2009), 1–8.

[46] C. Ó'Dúnlaing and C. K. Yap. 1985. A 'retraction' method for planning the motion of a disc. *J. Algor.* 6, 1 (1985), 104–111.

[47] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. 2000. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams* (2nd ed.). John Wiley & Sons.

[48] R. Oliva and N. Pelechano. 2013. A generalized exact arbitrary clearance technique for navigation meshes. In *Proceedings of the 6th International Conference on Motion in Games*. 103–110.

[49] R. Oliva and N. Pelechano. 2013. NEOGEN: Near optimal generator of navigation meshes for 3D multi-layered environments. *Comput. Graph.* 37, 5 (2013), 403–412.

[50] S. Patil, J. P. van den Berg, S. Curtis, M. C. Lin, and D. Manocha. 2010. Directing crowd simulations using navigation fields. *IEEE Trans. Vis. Comput. Graph.* 17, 2 (2010), 244–254.

[51] J. Pettré, J.-P. Laumond, and D. Thalmann. 2005. A navigation graph for real-time crowd animation on multilayered and uneven terrain. In *Proceedings of the 1st International Workshop on Crowd Simulation*. 81–89.

[52] M. Pocchiola and G. Vegter. 1993. The visibility complex. In *Proceedings of the 9th Annual Symposium on Computational Geometry*. 328–337.

[53] R. M. Polak. 2016. *Extracting Walkable Areas from 3D Environments*. Master's Thesis. Utrecht University.

[54] F. Preparata. 1977. The medial axis of a simple polygon. In *Mathematical Foundations of Computer Science*. Vol. 53. Springer, 443–450.

[55] C. Reynolds. 1999. Steering behaviors for autonomous characters. In *Proceedings of the Game Developers Conference*. 763–782.

[56] B. C. Ricks and P. K. Egbert. 2014. A whole surface approach to crowd simulation on arbitrary topologies. *IEEE Trans. Vis. Comput. Graph.* 20, 2 (2014), 159–171.

[57] S. Rodriguez and N. M. Amato. 2011. Roadmap-based level clearing of buildings. In *Proceedings of the 4th International Conference on Motion in Games*. 340–352.

[58] M. I. Shamos and D. Hoey. 1975. Closest-point problems. In *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science*. 151–162.

[59] G. Snook. 2000. Simplified 3D movement and pathfinding using navigation meshes. In *Game Programming Gems*, Mark DeLoura (Ed.). Charles River Media, 288–304.

[60] N. Sturtevant. 2012. Benchmarks for grid-based pathfinding. *Trans. Comput. Intell. AI Games* 4, 2 (2012), 144–148.

[61] D. Thalmann and S. R. Musse. 2013. *Crowd Simulation* (2nd ed.). Springer.

[62] W. G. van Toll, A. F. Cook IV, and R. Geraerts. 2011. Navigation meshes for realistic multi-layered environments. In *Proceedings of the 24th IEEE/RSJ International Conference on Intelligent Robots and Systems*. 3526–3532.

[63] W. G. van Toll, A. F. Cook IV, and R. Geraerts. 2012. A navigation mesh for dynamic environments. *Comput. Anim. Virtual Worlds* 23, 6 (2012), 535–546.

[64] W. van Toll and R. Geraerts. 2015. Dynamically Pruned A* for re-planning in navigation meshes. In *Proceedings of the 28th IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2051–2057.

[65] W. van Toll, N. Jaklin, and R. Geraerts. 2015. Towards believable crowds: A generic multi-level framework for agent navigation. In *Proceedings of the ASCI.OPEN/ ICT.OPEN (ASCI track) Conference*.
[66] W. van Toll, R. Triesscheijn, M. Kallmann, R. Oliva, N. Pelechano, J. Pettré, and R. Geraerts. 2016. A comparative study of navigation meshes. In *Proceedings of the 9th ACM SIGGRAPH International Conference on Motion in Games*. 91–100.
[67] A. Treuille, S. Cooper, and Z. Popović. 2006. Continuum crowds. *ACM Trans. Graph.* 25 (2006), 1160–1168. Issue 3.
[68] Unity3D Game Engine. 2018. Retrieved from http://www.unity3d.com/.
[69] R. Wein, J. P. van den Berg, and D. Halperin. 2007. The visibility-Voronoi complex and its applications. *Comp. Geom. Theor. Appl.* 36 (2007), 66–87.
[70] E. Whiting, J. Battat, and S. Teller. 2007. Topology of urban environments: Graph construction from multi-building floor plan data. In *Proceedings of the 12th International Conference on Computer-Aided Architectural Design*. 115–128.
[71] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. 1999. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings of the 16th IEEE International Conference on Robotics and Automation*. 1024–1031.

Received June 2016; revised October 2017; accepted March 2018