Multi-Layered Navigation Meshes

Wouter G. van Toll, Atlas F. Cook IV, Roland Geraerts

Abstract

Virtual characters often need to plan visually convincing paths through a complicated environment. For example, a traveler may need to walk from an airport entrance to a staircase, descend the staircase, walk to a shuttle, ride the shuttle to a destination, ride an elevator back to the ground floor, and finally move on the ground floor again to reach the desired airplane. Most previous research only supports path planning in a single plane because the underlying data structures are two-dimensional. The goal of this paper is to permit visually convincing paths to be efficiently computed in a multi-layered environment such as an airport or a multi-storey building. We describe an algorithm to create a navigation mesh, and our implementation demonstrates the feasibility of the approach.

A multi-layered environment is represented by a set of two-dimensional layers and a set of connections. Each layer is a collection of two-dimensional polygons that all lie in a single plane, and each connection provides a means of moving between layers.

We first compute the traditional medial axis of each two-dimensional layer in the environment. The connections are then used to iteratively merge this collection of medial axes into a single data structure. By adding a linear number of line segments that connect the medial axis to the nearest obstacles, we obtain a navigation mesh that mathematically describes the walkable areas in a multi-layered environment. This mesh can easily be input into existing planners to generate visually convincing paths for thousands of characters in real-time.

1 Introduction

Robots and virtual characters frequently need to plan visually convincing paths through a complicated environment. Such paths should be easy to compute and should permit virtual characters to avoid static obstacles as well as other moving virtual characters. The goal of this paper is to permit realistic paths to be efficiently computed in a non-planar environment such as a multi-storey building or an airport.

A common strategy for efficiently computing realistic paths is to partition the environment into a collection of walkable areas. This partition is often referred to as a *navigation mesh*. A useful data structure that can be used to construct a navigation mesh is the medial axis. The *medial axis* is the set of all points in an environment that have more than one distinct closest point on the boundary of the environment [21].

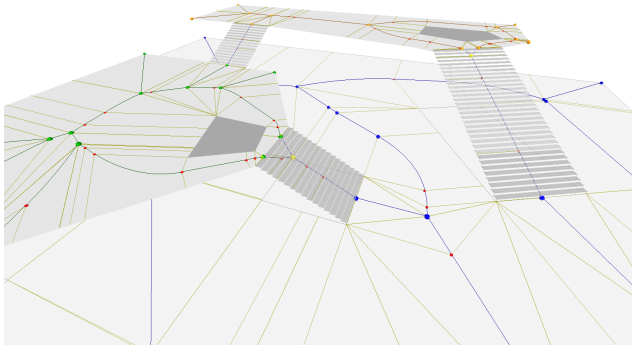


Figure 1: A multi-layered environment with three layers (ground floor, first floor, and second floor). Here, staircases provide a means of moving between layers. Obstacles are darkly-shaded. Our navigation mesh is constructed by augmenting a medial axis with line segments that connect this medial axis to the nearest obstacle. Large disks depict vertices on the medial axis. Small disks illustrate points on the medial axis where a nearest obstacle changes.

We initially compute the traditional medial axis of each two-dimensional walkable region in the environment. This collection of medial axes is then merged together into a single multi-layered medial axis based on the connections in the environment. By augmenting this structure with additional line segments that connect the medial axis to the nearest obstacles, we obtain a partition of the multi-layered environment into a set of walkable areas. These walkable areas define a navigation mesh such that all points in a walkable area have the same nearest obstacle(s). We refer the reader to Figure 1 for an example.

1.1 Related Work in Two-Dimensions

Consider a two-dimensional polygonal environment that contains n vertices. If the goal is simply to return a *shortest* path through this two-dimensional environment, then the following exact approaches exist.

Given one fixed source point s in a two-dimensional

This work was partially supported by the ITEA2 Metaverse1 (www.metaverse1.org) Project, the GATE project, and by INCONTROL Simulation Solutions. The authors are part of the Institute of Information and Computing Sciences, Utrecht University, 3584 CC Utrecht, the Netherlands. E-mail addresses: wouter@vantoll.nl, A.F.CookIV@uu.nl, R.J.Geraerts@uu.nl.

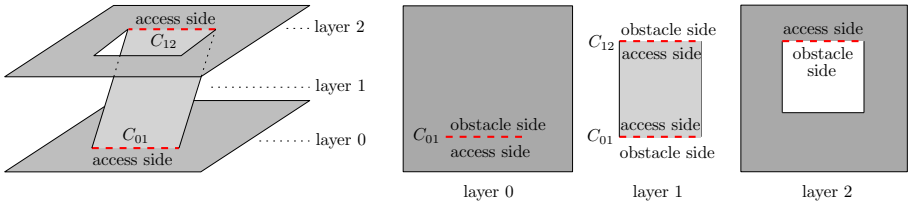


Figure 2: A multi-layered environment in $\mathbb{R}^3$ can be represented as a set of two-dimensional layers and a set of connections between these layers. For example, the connection C_{01} connects layer 0 and layer 1, and the connection C_{12} connects layer 1 and layer 2. Each connection is directed in the sense that it can only be used through its access side. The obstacle side of a connection is an impassable obstacle.

polygonal environment, Hershberger and Suri [9] show how to quickly compute any shortest path from the source point s. Their $O(n \log n)$ preprocessing step uses the continuous Dijkstra paradigm to maintain the "wavefront" of all points in the environment that are equidistant to s. The main drawback to this approach is that their preprocessing technique is inherently only relevant for one *fixed* source point.

Shortest paths can also be computed between *any* pair of points in a two-dimensional environment with a visibility graph. A visibility graph draws an edge between all pairs of mutually visible vertices in the environment. Such a visibility graph can be constructed in $O(n^2)$ time [7] and can be combined with A* search [8] or Dijkstra's algorithm [3] to quickly return a shortest path between any two points in a two-dimensional environment.

Although the above approaches can be used to compute paths through a two-dimensional polygonal environment, *shortest* paths always turn at obstacle vertices. This behavior causes a virtual character to nearly collide with many obstacles and to make many sharp turns. As a result, recent research has moved toward graph-based techniques and navigation meshes.

Graph-based techniques such as probabilistic roadmaps [16], rapidly-exploring random trees [17], and waypoint graphs [22] represent the environment using a set of *one-dimensional* edges. By contrast, a *navigation mesh* partitions the environment into walkable areas that are *two-dimensional*. These walkable areas permit virtual characters to control their movements inside each two-dimensional region [14]. This flexibility also makes it much easier for virtual characters to avoid other moving virtual characters.

There are many popular approaches for partitioning a two-dimensional polygonal environment into walkable areas. Wein et al. [24] show how to construct a Visibility Voronoi Complex in $O(n^2 \log n)$ time. Their approach combines visibility information with the two-dimensional areas of a Voronoi diagram. The structure also encodes *clearance information* that describes the nearest obstacle to each point in the environment. Although their technique is designed for two-dimensional polygonal environments, it could be extended to a multi-layered environment by using our new algorithm.

Kallmann [12] uses a special *triangulation* to construct a navigation mesh in $O(n \log n)$ time. The amount of clearance along a path in this triangulation is based on the radius of the largest empty disk along the path. Such a triangulation has linear complexity and encodes clearance information for many points in the environment.

A linear-sized medial axis can also be used to encode clearance information for all points in the environment. Such a medial axis is the set of all points in the environment that have more than one distinct closest point on the boundary of the environment [21]. Geraerts [5] uses a special type of medial axis called the Explicit Corridor Map to partition a two-dimensional environment into walkable areas in $O(n \log n)$ time.

An advantage of navigation meshes that encode clearance information is that A* search [8] can quickly determine a reasonably short path that for the most part follows the edges of the navigation mesh yet still manages to stay far away from obstacles. Force-based approaches can then be used to ensure that this path is smooth and visually convincing [13]. Local schemes can also ensure that virtual characters make small course adjustments when they avoid other moving entities [2, 14, 15]. Such techniques have been used to successfully simulate thousands of moving characters in a two-dimensional environment [2, 6].

1.2 Related Work in a Multi-Layered Environment

The goal of this paper is to permit realistic paths to be efficiently computed in a non-planar environment that cannot be described in two dimensions. Specifically, we consider an environment such as a multi-storey building or an airport that can be described by a set of two-dimensional polygons.

As illustrated in Figure 2, a *multi-layered environment* is composed of a set of two-dimensional polygonal *layers* and a set of *connections* between these layers. A layer is a set of two-dimensional polygons that all lie in the same plane. Thus, a layer typically represents one floor of a building. A connection is a line segment that provides a means of moving from one layer to another layer. Each connection is directed in the sense that it has two sides: the *access side* allows movement between layers, and the *obstacle side* is an impassable obstacle.

Throughout this paper, we assume that a multi-layered environment is *realistic* and *walkable*. A realistic environment is one that could be constructed in the real world; thus, a con-

nection cannot magically transport a virtual character to an arbitrary location. A walkable environment is one in which a human could successfully traverse; hence, a virtual character should never walk on a very steep layer and should never defy gravity by walking along the bottom of a layer.

Several researchers have studied how a multi-layered environment should be represented in order to best facilitate high-quality path planning. Tsetsos et al. [23] describe a semantic model for indoor scenes that encodes junctions, room exits, and corridor sections. Jiang et al. [11] describe a multi-layered environment as a set of interconnected objects plus a height component for staircases and ramps.

Deusdado et al. [4] show how to decompose an existing polygonal environment into a set of layers and connections, and Whiting et al. [25] show how to extract a graph from a multi-layered CAD drawing. Since these automated decomposition techniques exist, we will assume throughout this paper that the given multi-layered environment has already been decomposed into layers and connections.

Existing path planning techniques for a multi-layered environment rely on approximations and parameter tuning. Pettré et al. [20] use a set of overlapping disks to describe the walkable space. Most software packages such as Mononen's [18] open-source *Recast Navigation* project work similarly. The software typically discretizes the world into cubic voxels, extracts the walkable surfaces, and connects all adjacent cells.

Although the above approaches can often be fine-tuned to produce good results for a given environment, none of them provides an exact and compact description of the walkable space. The main advantage of our technique is that we produce a mathematical description of all walkable surfaces without requiring parameter tuning or voxel-like approximations. Note that we do not currently include a height component with each layer because we assume that a realistic multi-layered environment will not contain any layers that are very steep. We also assume that the environment will not contain any low ceilings that would give a headache to a very tall virtual character. For now, such scenarios could be explicitly modeled by placing an obstacle below a low ceiling. In the future, we plan to add height information to the mesh.

1.3 Results

Given a multi-layered environment that contains n vertices and k connections, this paper shows how to construct a multi-layered data structure in $O(kn \log n)$ time and $O(kn)$ space. Our multi-layered data structure combines the benefits of both a medial axis and a navigation mesh. Like a medial axis, the data structure produces high-clearance paths because it encodes the nearest obstacle(s) to every point in the multi-layered environment. Like a navigation mesh, the data structure partitions the environment into a set of walkable areas. The result is a simple navigation mesh that mathematically describes all walkable surfaces. This data structure can easily be used by existing path planners such as the Indicative Route Method [14] and the Reciprocal Velocity Obstacles technique [2] to generate visually convincing paths for thousands of virtual characters in real-time.

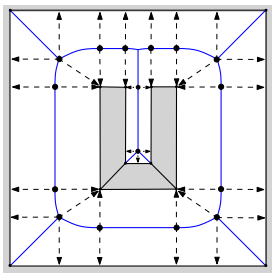


Figure 3: The medial axis of a two-dimensional environment is shown in blue. Obstacles are shown in gray. The dashed line segments make it easy to determine the nearest obstacle in the environment.

The remainder of this paper is organized as follows. In Section 2, we formally describe the medial axis and review the traditional methods of computing this structure. Section 3 contains an algorithm to compute a medial axis in a multi-layered environment. Section 4 discusses an implementation that uses this medial axis to compute visually convincing paths through a multi-layered environment. Section 5 describes some experiments that demonstrate the feasibility of our approach.

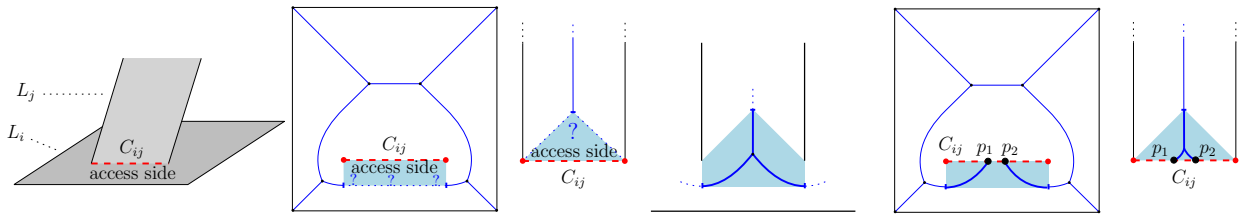
2 Preliminaries

The *medial axis* is the set of all points in an environment that have more than one distinct closest point on the boundary of the environment [21]. Figure 3 illustrates the medial axis of a two-dimensional environment that contains a U-shaped obstacle and a bounding square obstacle.

The medial axis of a two-dimensional environment with n vertices can be computed *exactly* in $O(n \log n)$ time and $O(n)$ space using plane sweep techniques [3]. Alternatively, the medial axis can be *approximated* by using graphics hardware to project distance functions onto an orthogonal plane [10]. Note that the projection technique inherently requires an environment that is planar because it requires that the entire traversable space is visible from above.

The medial axis has the same complexity as a traditional triangulation, but it contains more information. In particular, a medial axis encodes clearance information for every point in the environment. This permits the efficient computation of minimum clearance paths that stay far away from obstacles [5].

The medial axis is also well-defined for three-dimensional environments. However, it is extremely difficult to mathematically describe and implement the surface patches that define this structure. Consequently, it is impractical to compute the medial axis in three-dimensional environments [10].



(a) Multi-layered environment (b) Partial medial axes for L_i and L_j (c) Connection scene S_{ij} (d) Updated medial axes for L_i and L_j

Figure 4: The medial axis of a simple multi-layered environment. The influence zone Z_{ij} is highlighted in blue. The intersection of M_i and M_j with C_{ij} is denoted by the points $p_1, p_2 \in C_{ij}$.

3 Computing the Medial Axis of a Multi-Layered Environment

A multi-layered environment E consists of a set $\mathsf{L} = \{L_1, ..., L_x\}$ of two-dimensional layers and a set C of connections. Throughout this paper, a connection $C_{ij} \in \mathsf{C}$ always denotes a line segment that connects two layers $L_i \in \mathsf{L}$ and $L_j \in \mathsf{L}$. Please refer to Figure 4a.

Let M_E denote the medial axis of the multi-layered environment E. Since we are interested in using M_E to compute paths that are entirely contained on the surfaces in L, we define the medial axis M_E as the set of all points in L that have more than one closest point on the boundary of E. The main goal of this paper is to show how to compute the medial axis M_E.

At a high-level, M_E is computed as follows. The traditional two-dimensional medial axis is initially constructed for each layer in L under the assumption that every connection is an impassable obstacle. Each connection $C_{ij} \in \mathsf{C}$ is then iteratively *opened* so that paths can start to pass through it. When this occurs, we update the affected medial axes M_i and M_j to account for the new paths. Once all the connections have been opened, the medial axes in L will collectively encode the medial axis M_E for the multi-layered environment.

3.1 Opening a Connection

The process of opening a connection $C_{ij} \in \mathsf{C}$ will now be described in detail. Consider two layers L_i and L_j that are connected by a line segment connection C_{ij} as in Figure 4a. Let M_i be the two-dimensional medial axis of layer $L_i \in \mathsf{L}$ under the temporary assumption that the connection C_{ij} is an impassable obstacle.

The following definitions will be useful. The *influence zone* Z_{ij} is the set of all points in the multi-layered environment that have an interior point of the access side of C_{ij} as a nearest obstacle. The boundary of the influence zone consists of medial axis edges plus orthogonal line segments that connect these medial axis edges to the endpoints of the connection. The *connection scene* S_{ij} is the set of all obstacles (excluding the access side of C_{ij}) that are closest to at least one point in the influence zone Z_{ij}. Note that the medial axes $M_1, ..., M_x$ for the layers in L encode exactly the information that is needed to determine both the influence zone Z_{ij} and the connection scene S_{ij}.

When a connection C_{ij} is opened, paths can begin to pass through the access side of C_{ij}. This means that we are effectively removing the obstacle from the environment that corresponds to the access side of C_{ij}. Consequently, we will need to determine a new closest obstacle for all points that were previously closest to the access side of C_{ij}. All of these points are by definition in the influence zone Z_{ij}. No other points need to have their nearest obstacle information updated. This follows because points that are not in Z_{ij} must be closer to some other obstacle than to C_{ij}; thus, no obstacle that could be reached by passing through C_{ij} could possibly define a nearest obstacle. The above arguments immediately imply the following lemma:

Lemma 1 *When a connection C_{ij} is opened, only the points in the influence zone Z_{ij} need to have their nearest obstacle information updated.*

The next task is to determine how to update the closest obstacle information for the points in the influence zone Z_{ij}. The below lemma proves that a nearest obstacle for every point in Z_{ij} must be in the connection scene S_{ij}.

Lemma 2 *A nearest obstacle for every point in the influence zone Z_{ij} is in the connection scene S_{ij}.*

Proof. An obstacle that is not in the connection scene S_{ij} is by definition not optimally close to any point on C_{ij}. Therefore, this obstacle will never be optimally close to any point that can only be reached by passing through C_{ij}. □

Lemma 1 ensures that when a connection C_{ij} is opened, we only need to update the edges of the medial axes $M_1, ..., M_x$ that are inside the influence zone Z_{ij}. Lemma 2 guarantees that these edges can be updated solely based on the obstacles in the connection scene S_{ij}.

Figure 4 illustrates the process of constructing the medial axis of a multi-layered environment that contains two layers L_i, L_j and one connection C_{ij}. The medial axes M_i and M_j are initially constructed under the assumption that C_{ij} is an impassable obstacle. When the connection C_{ij} is eventually

opened so that paths can pass through its access side, we only need to update the medial axes edges inside the highlighted influence zone Z_{ij} (see Figure 4b).

The connection scene S_{ij} is the set of all obstacle points in the environment that are closest to the access side of C_{ij} (see Figure 4c). Since we want to use a standard two-dimensional algorithm to update the medial axes edges in the influence zone, we first need to place all of the obstacle points in the connection scene S_{ij} into a common plane. This can be achieved (in a heuristic sense) by projecting all of the points in S_{ij} onto the plane supporting the layer L_i.[1]

Once all of the obstacles in the connection scene S_{ij} lie in a common plane, the medial axis of the two-dimensional connection scene can be computed using a traditional two-dimensional algorithm. The resulting medial axis precisely defines all of the medial axis edges in the influence zone Z_{ij} (see Figure 4c). Inserting these edges into the affected medial axes will logically merge these structures together based on their common intersections with the line segment C_{ij} (see Figure 4d).

An iterative algorithm is now described that can construct the medial axis M_E for any multi-layered environment E.

3.2 Algorithm

1. Construct the traditional two-dimensional medial axes $M_1, ..., M_x$ for each layer $L_1, ..., L_x \in \mathsf{L}$ under the temporary assumption that every connection is an impassable obstacle.
2. For each connection C_{ij} in the environment E:
   - Determine the influence zone Z_{ij} and the connection scene S_{ij} using the medial axes $M_1, ..., M_x$.
   - Project all of the obstacles in the connection scene S_{ij} onto the plane supporting layer L_i. Let M_S be the two-dimensional medial axis of these projected obstacles.
   - Use the medial axis M_S to update the edges and vertices of $M_1, ..., M_x$ inside the influence zone S_{ij}.
   - Mark the connection C_{ij} as fully processed.

One advantage of the above iterative approach is that a single obstacle is correctly permitted to influence the medial axes of an arbitrary number of layers. For example, the bottom-left polygonal obstacle in Figure 5 influences the medial axis on three layers (including the stairs as a layer). The connections can also be processed in an arbitrary order because all affected medial axes are corrected each time a connection is processed.

[1] This technique is a heuristic because projective transformations do not preserve Euclidean distances. Alternatively, one could rotate all of the obstacles about a connection until they become coplanar [1, 19]. The reason we use a projective transformation is that a realistic environment should not contain any layer that is exceptionally steep. In such a setting, projected distances reasonably approximate Euclidean distances.

3.3 Complexity Analysis

We now analyze the complexity of the above algorithm.

Lemma 3 *The medial axis M_S for one connection scene has $O(n)$ complexity and can be constructed in $O(n \log n)$ time.*

Proof. Although the $O(n)$ obstacles in a connection scene may lie in many different layers, these obstacles will never overlap when we project them onto the plane supporting layer L_i. This follows because a realistic and walkable multi-layered environment must have a 360° turn in order for projected obstacles to overlap. However, some obstacle point along such a turn must always be closer than a point after the turn. This means that the medial axis M_S is constructed from $O(n)$ disjoint obstacles. It has $O(n)$ complexity and can be constructed in $O(n \log n)$ time [3]. □

We can now state our main result.

Theorem 4 *The medial axis of a multi-layered environment with n vertices and k connections has $O(kn)$ complexity and can be constructed in $O(kn \log n)$ time.*

Proof. Construct the medial axes $M_1, ..., M_x$ for each of the two-dimensional layers $\{L_1, ..., L_x\} \in \mathsf{L}$ under the temporary assumption that all connections are impassable obstacles. This can easily be achieved in $O(n \log n)$ total time and $O(n)$ total space using traditional two-dimensional techniques.

For each connection C_{ij}, use the medial axes $M_1, ..., M_x$ to determine the influence zone Z_{ij} and the connection scene S_{ij} in $O(n)$ time. Project all of the obstacles in S_{ij} onto the plane supported by layer L_i. Compute the medial axis M_S for these obstacles in $O(n \log n)$ time as described in Lemma 3. The edges in M_S can then easily be used to update all of the edges of $M_1, ..., M_x$ inside the influence zone. Note that the intersections of these edges with C_{ij} provide a natural means of merging these medial axes together into a single structure.

Processing all connections in this fashion is sufficient to construct the medial axis of a multi-layered environment. Each connection takes $O(n \log n)$ time and $O(n)$ space to process by Lemma 3, so all k connections can be processed in $O(kn \log n)$ total time.

As illustrated in Figure 4d, an edge of the medial axis can intersect a connection. If we add a vertex at each such intersection point (so that each edge is associated with a single layer), then the total space requirement for the medial axis of a multi-layered environment is $O(kn)$. This follows because each of the $O(n)$ edges of the medial axis could intersect $O(k)$ line segment connections. □

4 Implementation

We have implemented an algorithm to compute the medial axis M_E of a multi-layered environment E. Each two-

dimensional medial axis is computed by using graphics hardware to project distance functions onto an orthogonal plane as in [5] and [10].

By annotating each edge in the medial axis with its defining layer and with closest obstacle information [5], we partition the multi-layered environment E into a set of two-dimensional walkable areas. These two-dimensional areas encode a navigation mesh that can be used to construct a visually convincing path between any two query points $s, g \in E$ as follows. Retract s and g onto nearby points on the medial axis M_E. Using A* search [8], calculate a *backbone path* through the medial axis that connects these two points [6]. The walkable areas in the environment that are adjacent to this backbone path define an *explicit corridor* [5]. Force-based steering techniques such as the Indicative Route Method [14] and Reciprocal Velocity Obstacles [2] can then be used to very quickly steer a virtual character through this corridor while avoiding obstacles and other moving virtual characters. Figure 5 illustrates that this technique can be used to construct a corridor that can guide a virtual character along a visually convincing path in a multi-layered environment.

5 Experiments

In this section, we will describe the experiments we have conducted on multi-layered navigation meshes. All of the experiments were performed on a PC with an NVIDIA GT 240 graphics card and an Intel Core2 Duo CPU (3.0 GHz) with 4 GB memory. Only one core was used. The application was implemented in Visual C++ under Windows 7.

The multi-layered navigation mesh in Figures 1 and 5 was computed in 46ms. The original environment has 230 obstacle vertices distributed over 8 connections and 7 layers (three floors and four staircases). The navigation mesh has 64 edges and 212 pairs of closest points.

Figure 6 shows the same 7 layers and 8 connections as in Figure 5; however, 53 additional obstacles have been added. The total number of obstacle vertices in this environment is 637. The navigation mesh was computed in 58ms. This mesh has 280 edges and 1084 pairs of closest points.

Figure 7 illustrates what happens when the number of connections is varied in a multi-layered environment. Figure 7a shows our navigation mesh when only 1 connection exists between two layers. Figure 7b illustrates the mesh when 5 connections are available. Figure 7c shows the mesh when 28 connections exist. Table 1 illustrates the time to compute this navigation mesh when various numbers of connections are present. The number of edges and pairs of closest points in the mesh are also reported. These experiments show that our technique is fast and scales well when obstacles or connections are added to an environment.

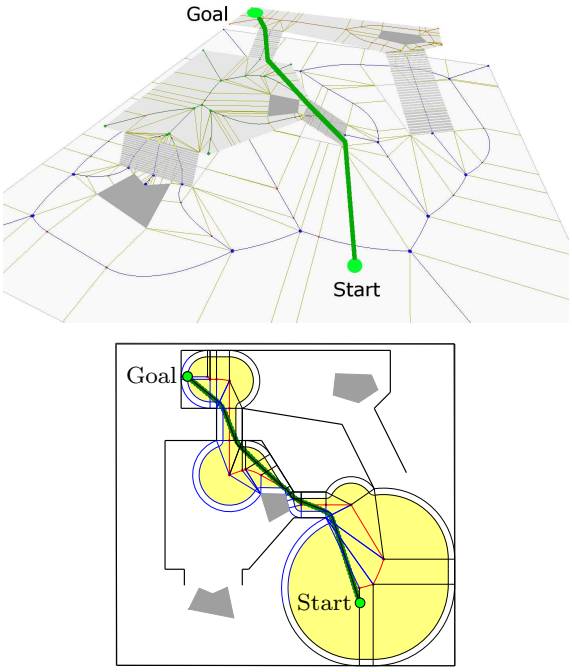


Figure 5: Our navigation mesh can be input into an existing path planner such as the Corridor Map Method [5] to generate a visually convincing path through a multi-layered environment. The above figures illustrate 3D and 2D views of the same multi-layered environment.

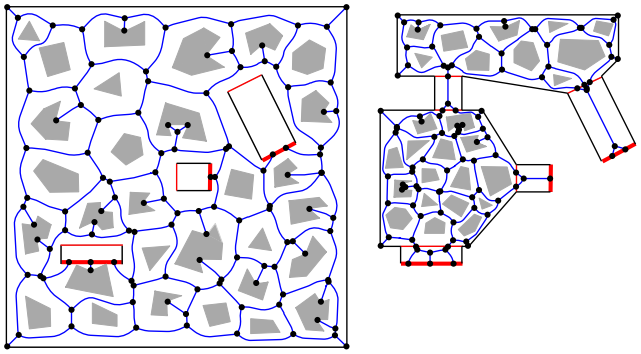


Figure 6: Many obstacles can be added to each layer.

6 Conclusion

Many modern virtual environments can be described by a set of two-dimensional polygonal layers and a set of connections between these layers. In this paper, we have proposed and implemented an algorithm to compute the medial axis in such an environment. The algorithm is simple and fast because it first computes a two-dimensional medial axis of each two-dimensional layer. Next, all of these medial axes are iteratively merged together into a single structure. By annotating edges in this merged structure with nearest obstacle information, a navigation mesh is obtained for the multi-layered environment. Such a navigation mesh can be input into existing path planners to generate visually convincing

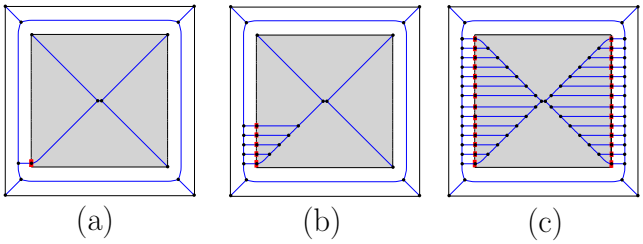


Figure 7: Many connections can exist between a pair of layers. The first layer is the area between the outer and inner rectangles. The second layer is the area inside the inner rectangle.

Table 1: Varying numbers of connections

Connections	Time	Mesh Edges, Pairs of Closest Points
1	7ms	15, 42
2	9ms	19, 55
3	11ms	23, 68
5	14ms	31, 94
8	19ms	43, 133
14	30ms	66, 208
20	43ms	88, 281
28	60ms	119, 382

paths for thousands of virtual characters in real-time.

As future work, the multi-layered data structure should be extended to support dynamic updates in a local and efficient manner. Dynamic updates would allow modeling connections that are only accessible at certain times (e.g., the doors of most buildings are locked at the end of each business day). Dynamic updates would also permit routes to be locally updated each time a new road/building was created.

In addition to dynamic updates, it would be interesting to add height information to the layers so that they could more accurately encode the three-dimensional structure of an environment. Finally, it sometimes makes more sense to hop over small obstacles than to go around them. For example, a hiker might choose to jump over holes and puddles along a rugged trail. These powerful extensions provide both a theoretical and practical basis for efficiently computing highly realistic paths in a multi-layered environment.

References

[1] P.K. Agarwal, B. Aronov, J. O'Rourke, and C.A. Schevon. Star unfolding of a polytope with applications. *SIAM Journal on Computing*, 26(6):1689–1713, 1997.

[2] J.P. van den Berg, M.C. Lin, and D. Manocha. Reciprocal Velocity Obstacles for real-time multi-agent navigation. *IEEE International Conference on Robotics and Automation*, pages 1928–1935, 2008.

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

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

[5] R. Geraerts. Planning short paths with clearance using Explicit Corridors. *IEEE International Conference on Robotics and Automation*, pages 1997–2004, 2010.

[6] R. Geraerts and M.H. Overmars. Enhancing corridor maps for real-time path planning in virtual environments. *Computer Animation and Social Agents*, pages 64–71, 2008.

[7] S.K. Ghosh and D.M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991.

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

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

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

[11] H. Jiang, W. Xu, T. Mao, C. Li, S. Xia, and Z. Wang. A semantic environment model for crowd simulation in multilayered complex environment. *16th ACM Symposium on Virtual Reality Software and Technology*, pages 191–198, 2009.

[12] M. Kallmann. Path planning in triangulations. *IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 49–54, 2005.

[13] I. Karamouzas, R. Geraerts, and M.H. Overmars. Indicative routes for path planning and crowd simulation. *4th International Conference on Foundations of Digital Games*, pages 113–120, 2009.

[14] I. Karamouzas, P. Heil, P. van Beek, and M.H. Overmars. A predictive collision avoidance model for pedestrian simulation. *2nd International Workshop on Motion in Games*, pages 41–52, 2009.

[15] I. Karamouzas and M.H. Overmars. A velocity-based approach for simulating human collision avoidance. *Intelligent Virtual Agents*, 6356:180–186, 2010.

[16] L.E. Kavraki, P. Švestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[17] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. *IEEE International Conference on Robotics and Automation*, pages 995–1001, 2000.

[18] M. Mononen. Recast navigation. *Google Project: http://code.google.com/p/recastnavigation*, 2011.

[19] D.M. Mount. The number of shortest paths on the surface of a polyhedron. *SIAM Journal on Computing*, 19(4):593–611, 1990.

[20] J. Pettré, J.P. Laumond, and D. Thalmann. A navigation graph for real-time crowd animation on multilayered and uneven terrain. *1st International Workshop on Crowd Simulation*, pages 1–9, 2005.

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

[22] S. Rabin. AI game programming wisdom 2. *Charles River Media Inc., Hingham*, 2004.

[23] V. Tsetsos, C. Anagnostopoulos, P. Kikiras, P. Hasiotis, and S. Hadjiefthymiades. A human-centered semantic navigation system for indoor environments. *International Conference on Pervasive Services*, pages 146–155, 2005.

[24] R. Wein, J.P. van den Berg, and D. Halperin. The Visibility-Voronoi Complex and its applications. *Computational Geometry: Theory and Applications*, 36(1):66–78, 2007.

[25] E. Whiting, J. Battat, and S. Teller. Topology of urban environments: Graph construction from multi-building floor plan data. *12th International Conference on Computer-Aided Architectural Design*, pages 115–128, 2007.