The Explicit Corridor Map: Using the Medial Axis for Real-Time Path Planning and Crowd Simulation

Wouter van Toll[1], Atlas F. Cook IV[2], Marc J. van Kreveld[1], and Roland Geraerts[1]

1 **Department of Information and Computing Sciences, Utrecht University;** `W.G.vanToll@uu.nl, M.J.vanKreveld@uu.nl, R.J.Geraerts@uu.nl`

2 **Information and Computer Sciences Department, University of Hawaii at Manoa;** `acook4@hawaii.edu`

Abstract

We describe and demonstrate the Explicit Corridor Map (ECM), a *navigation mesh* for path planning and crowd simulation in virtual environments. For a bounded 2D environment with polygonal obstacles, the ECM is the *medial axis* of the free space annotated with nearest-obstacle information. It can be used to compute short and smooth paths for disk-shaped characters of any radius. It is also well-defined for multi-layered 3D environments that consist of connected planar layers. We highlight various operations on the ECM, such as dynamic updates, visibility queries, and the computation of paths (*indicative routes*).

We have implemented the ECM as the basis of a real-time *crowd simulation framework* with path following and collision avoidance. Our implementation has been successfully used to simulate real-life events involving large crowds of heterogeneous characters. The enclosed *demo application* displays various features of our software.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling – I.6.8 Types of Simulation

Keywords and phrases Medial axis, Navigation mesh, Path planning, Crowd simulation

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.70

1 Introduction

Many simulations, games, and other applications feature virtual characters that need to plan and traverse paths through a complicated environment in real-time. Our research focuses on *path planning and crowd simulation* for disk-shaped characters that move along walkable surfaces. A *navigation mesh* subdivides these surfaces into regions for path planning purposes. We describe the Explicit Corridor Map (ECM) [1, 8], a navigation mesh that allows path planning for characters of an arbitrary radius. This document describes the theoretical background of our demo application, as well as a number of implementation details. Various parts of this text have been extracted from a journal article that is currently in review.

2 Definitions

Let a *2D environment* $\mathcal{E}$ be a bounded two-dimensional planar environment with polygonal obstacles. The *obstacle space* $\mathcal{E}_{obs}$ is the union of all obstacles, including the boundary of the environment. The complement of $\mathcal{E}_{obs}$ is the *free space* $\mathcal{E}_{free}$. The *complexity* of $\mathcal{E}$ is the number of vertices n required to define $\mathcal{E}_{obs}$ using interior-disjoint simple polygons.

The Explicit Corridor Map (ECM) is based on the *medial axis* (MA) [6], which is related to the Voronoi diagram of line segment sites [5]. Various definitions of the MA exist; we define it as the closure of all points in $\mathcal{E}_{free}$ that have at least two distinct equidistant nearest points in $\mathcal{E}_{obs}$, in terms of 2D Euclidean distance. An example is shown in Figure 1a.

The ECM is an MA annotated with nearest-obstacle information. More precisely, it is an undirected graph $G = (V, E)$ where V is the set of MA vertices of degree 1, 3, or higher. Each edge $e_{ij} \in E$ is a sequence of MA arcs between two vertices v_i and v_j in V. An edge is represented by $n' \geq 2$ *bending points* $bp_0, \ldots, bp_{n'-1}$ where $bp_0 = v_i$, $bp_{n'-1} = v_j$, and $bp_1, \ldots, bp_{n'-2}$ are the degree-2 MA vertices inbetween. Each 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. An example of an ECM is shown in Figure 1b.

The bending points and their annotations induce a subdivision of $\mathcal{E}_{free}$ into polygonal *ECM cells*. Therefore, the ECM serves as a navigation mesh. Similarly to the medial axis, the ECM can be constructed in $\mathcal{O}(n \log n)$ time and requires $\mathcal{O}(n)$ space.

We have extended the MA and ECM to *multi-layered environments* that consist of 2D layers connected by k line segment connections. The MA is based on distances projected onto a common ground plane. The multi-layered ECM can be computed in $\mathcal{O}(kn \log n)$ time by initially treating the connections as obstacles and then opening them incrementally [8]. An article with improved definitions, proofs, and algorithms is currently under submission.

3 Operations

Given a query point q, we can find the ECM cell that contains q in $\mathcal{O}(\log n)$ time using a standard point location data structure. Given this cell, it takes $\mathcal{O}(1)$ time to compute the *nearest obstacle point* n_q to q and the retraction $Retr(q)$ of q, which is the point where the half-line from n_q through q first intersects the medial axis.

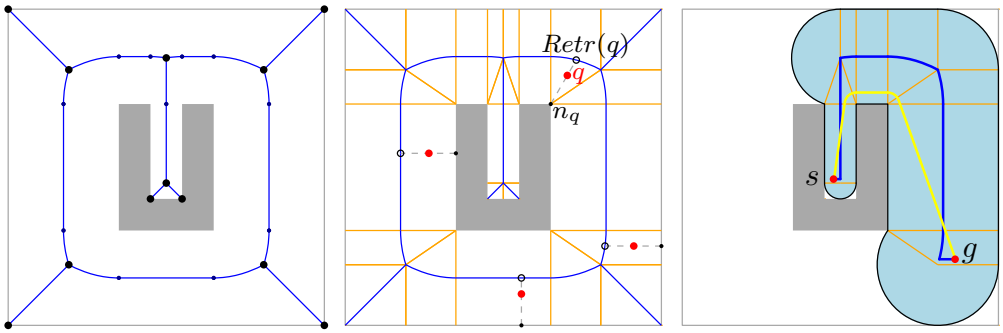
When an obstacle is inserted or deleted during the simulation, the ECM can be *dynamically updated* in real-time [9]. Our algorithms for dynamic updates are based on algorithms for site insertions and deletions in Voronoi diagrams. Their running times depend on the number of neighboring obstacles for the dynamic obstacle.

We can also use the ECM to efficiently compute the *visibility polygon* of a query point, or to check if two points are *mutually visible*. These algorithms automatically work in multi-layered environments because they only rely on the connectivity between ECM cells.

4 Path Planning and Crowd Simulation

To compute a global path from a point s to a point g, we retract the query points and then use A* search [2] to find a shortest path from $Retr(s)$ to $Retr(g)$ on the medial axis. Because the ECM stores nearest-obstacle information, we can dynamically ignore edges that are too narrow for a character to use. Therefore, we can use the ECM to plan paths for characters of an arbitrary radius. The resulting medial axis path induces a *corridor* of free space around the medial axis. Within this corridor, we can compute various types of geometric paths, such as a shortest path with clearance to obstacles [1]. Figure 1c summarizes this.

A geometric path can be used as an *indicative route* for the character to follow smoothly during the simulation. In each simulation step, the character computes a *preferred* velocity that steers it further along the indicative route. Next, the character converts this preferred velocity to an *actual* velocity that avoids collisions with other characters. For more details on these simulation components, we refer the reader to an overview paper [7].



(a) Medial axis **(b)** ECM / Retractions **(c)** Path planning

Figure 1 The ECM and its operations in a simple 2D environment. a Obstacles are shown in gray. The medial axis is shown in blue. Degree-2 vertices are shown as small dots; other vertices are shown as large dots. b The ECM adds nearest-obstacle annotations, shown as orange line segments. For any query point q (red dot), we can use the ECM to compute the nearest obstacle point n_q (black dot) and the retraction $Retr(q)$ (circle). c To plan a path from s to g, we first compute a shortest path on the medial axis. This induces a corridor of free space (light blue) in which we can compute e.g. a short route with clearance to obstacles (shown in yellow).

5 Implementation

We have implemented a framework that combines the ECM with algorithms for path planning, path following, collision avoidance, and other tasks related to crowd simulation. The software was written in C++ in Visual Studio 2013, but the code is platform-independent.

5.1 Computing the ECM

To robustly compute the ECM, we have integrated two different libraries for computing Voronoi diagrams of line segment sites: Vroni [3] and a package of Boost (`http://www.boost.org/`). First, we compute the Voronoi diagram using either library. Next, traverse it while adding nearest-obstacle annotations and removing vertices and edges that are not part of the medial axis. Finally, we remove all connected components that lie inside obstacles. For the multi-layered ECM, we first perform these steps for each layer separately (possibly in parallel), and then we open the connections one by one.

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 and comparative purposes, we use these rounded coordinates in Vroni as well. Since we use meters as units, this scaling implies that we represent all coordinates within a precision of 0.1 millimeters.

We have also created an approximating GPU-based construction algorithm [4, 1], but this approach cannot guarantee certain properties such as topological correctness, accurate positions of vertices, and full coverage of the free space. This makes many other ECM algorithms very difficult to implement, most notably our algorithms for dynamic updates and the construction algorithm of the multi-layered ECM.

Experiments show that our Vroni-based implementation is faster than the Boost-based version. However, the advantage of Boost is that it is thread-safe, which allows us to compute multi-layered ECMs very efficiently by computing the ECMs of all layers in parallel. Figure 2 shows an example of a huge multi-layered environment. Its ECM, with over 40,000 vertices and 120,000 bending points, was computed in under 14 seconds using Vroni.

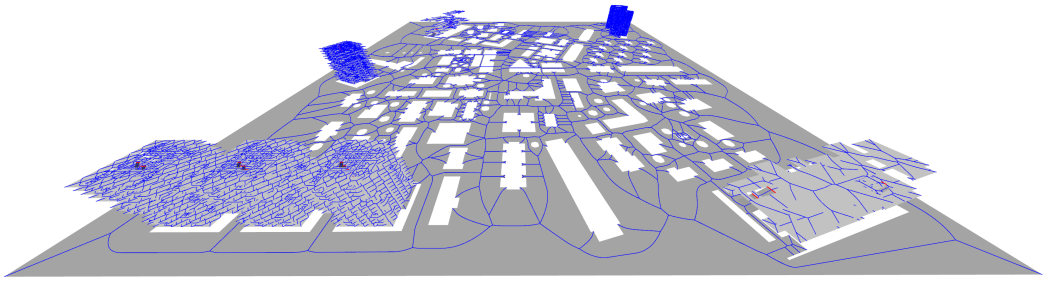


Figure 2 A large multi-layered city and its medial axis, computed using our ECM implementation.

5.2 Demo Application and Other Results

The enclosed demo application automatically loads four 2D environments and computes their ECMs using Boost. In the first environment, it creates a character at a random position and lets it compute and follow an indicative route to a random goal position. Whenever the character reaches its goal, it computes a new route to a new random position. The user can switch between environments, assign new goal positions, change the radius of the character (which may affect the accessibility of certain ECM edges), and change the preferred distance to obstacles (which affects the indicative route, but not the corridor in which it lies).

The full version of our framework can simulate tens of thousands of heterogeneous characters in real-time using multi-threading and an efficient subdivision of the simulation loop into substeps [7]. The framework can also be linked as a DLL to other software, such as the Unity3D game engine (`http://www.unity3d.com/`).

References

1 R. Geraerts. Planning short paths with clearance using Explicit Corridors. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1997–2004, 2010.

2 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.

3 M. Held. 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*, pages 3–12, 2011.

4 K.E. Hoff III, T. Culver, J. Keyser, M. 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.

5 D.T. Lee and R.L. Drysdale III. Generalization of Voronoi diagrams in the plane. *SIAM Journal on Computing*, 10(1):73–87, 1981.

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

7 W. van Toll, N. Jaklin, and R. Geraerts. Towards believable crowds: A generic multi-level framework for agent navigation. In *ASCI.OPEN*, 2015.

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

9 W.G. van Toll, A.F. Cook IV, and R. Geraerts. A navigation mesh for dynamic environments. *Computer Animation and Virtual Worlds*, 23(6):535–546, 2012.