Computers & Graphics 91 (2020) 52–82

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Comparing navigation meshes: Theoretical analysis and practical metrics ☆

Wouter van Toll[a,*], Roy Triesscheijn[f], Marcelo Kallmann[b], Ramon Oliva[c], Nuria Pelechano[d], Julien Pettré[a], Roland Geraerts[e]

[a] Univ Rennes, Inria, CNRS, IRISA, France
[b] University of California Merced, USA
[c] Universitat de Barcelona, Spain
[d] Universitat Politècnica de Catalunya, Spain
[e] Utrecht University, The Netherlands
[f] Roy Triesscheijn was previously a student at Utrecht University.

ARTICLE INFO

Article history:
Received 31 March 2020
Revised 27 May 2020
Accepted 20 June 2020
Available online 4 July 2020

Keywords:
Navigation meshes
Virtual environments
Geometry processing
Path planning

ABSTRACT

A navigation mesh is a representation of a 2D or 3D virtual environment that enables path planning and crowd simulation for walking characters. Various state-of-the-art navigation meshes exist, but there is no standardized way of evaluating or comparing them. Each implementation is in a different state of maturity, has been tested on different hardware, uses different example environments, and may have been designed with a different application in mind. In this paper, we develop and use a framework for comparing navigation meshes. First, we give general definitions of 2D and 3D environments and navigation meshes. Second, we propose theoretical properties by which navigation meshes can be classified. Third, we introduce metrics by which the quality of a navigation mesh implementation can be measured objectively. Fourth, we use these properties and metrics to compare various state-of-the-art navigation meshes in a range of 2D and 3D environments. Finally, we analyze our results to identify important topics for future research on navigation meshes. We expect that this work will set a new standard for the evaluation of navigation meshes, that it will help developers choose an appropriate navigation mesh for their application, and that it will steer future research in interesting directions.

1. Introduction

Path planning for moving characters in virtual environments is a fundamental task in simulations and games. Modern applications feature increasingly large crowds of characters; each character needs to autonomously compute and follow a path while avoiding collisions with obstacles and other characters. This leads to many queries related to e.g. path planning, path following, point location, and collision avoidance [1]. A crowd simulation is expected to run in real-time despite all these demands. This stresses the need for high-quality data structures and algorithms.

Path planning for characters is different from robot motion planning, in which the high-dimensional *configuration space* of a robot [2] is often represented as a sampling-based graph (e.g. [3,4]). In our domain, the environments are typically three-dimensional, but characters are constrained to surfaces that are sufficiently flat to walk on. The behavior of characters may also include crawling, running, and other surface-based movement, but we will speak of *walking* and *walkable surfaces* for simplicity. The walkable surfaces of an environment form the *free space* $\mathcal{E}_{free}$, which is usually less complex than the environment itself.

A *navigation mesh* is a representation of $\mathcal{E}_{free}$ as a set of (usually polygonal) regions, along with a graph that describes how these regions are connected. For path planning, characters first find a path in the graph and then compute a suitable geometric route through the corresponding regions. A research topic of increasing importance is the *automatic* construction of a navigation mesh for any input environment (an arbitrary collection of polygons in 3D). Current construction algorithms can roughly be placed into one of two categories: *voxel-based* algorithms that approximate the walkable surfaces from raw 3D geometry, or *exact* algorithms that require pre-processed input (e.g. a set of 2D layers) to compute a navigation mesh with a provable worst-case complexity. These two categories are difficult to compare, especially because voxel-based algorithms introduce a trade-off between accuracy and speed. Fur-

☆ This article was recommended for publication by R Boulic

* Corresponding author.
E-mail address: wouter.van-toll@inria.fr (W. van Toll).

https://doi.org/10.1016/j.cag.2020.06.006
0097-8493/

thermore, each method uses its own set of test environments to show its own (dis)advantages. To steer subsequent research into interesting directions, an objective comparison between navigation meshes is required.

Goals and contributions. In this paper, we conduct a comparative study of navigation mesh implementations by using the same hardware, quality metrics, and input environments for all methods. Our goal is to propose a way to objectively measure how suitable particular navigation meshes are for particular environments. Because navigation meshes have many applications with different requirements, it is difficult to propose a single criterion that can identify 'the best' navigation mesh. Instead, we present a collection of criteria, each of which is relevant for particular applications. Our main contributions are the following:

- We propose properties by which the data structures and algorithms of navigation meshes can be classified (Section 4), and we use them in a theoretical comparison of various state-of-the-art navigation meshes (Section 5).
- We present quantitative metrics to measure the quality and performance of a navigation mesh implementation for a given input environment (Section 6). In particular, we address the concept of coverage in 3D.
- We combine these metrics into a benchmark tool, and we use it to compare state-of-the-art navigation mesh implementations in a range of 2D and 3D environments (Section 7). This comparison identifies limitations of voxel-based methods in terms of connectivity preservation and scalability to large environments.
- We discuss our results to identify the most important topics for future work (Section 9), such as the comparison of path quality across navigation meshes, and the development of non-voxel-based algorithms for filtering a 3D environment.

We emphasize that our goal is not to expose which navigation mesh implementation is 'the best'. Instead, by presenting generic definitions and metrics and using one common test platform, we intend to set a standard for the analysis of navigation meshes, and to expose areas for future research.

This paper is an extension of our previous work [5] which reported partial results available at the time. In this extended version, we describe our full study. Specifically, we now include a more thorough theoretical comparison of navigation meshes (Section 5), a larger set of input environments (Section 7.3), a more careful consideration of parameter settings (Section 7.2), a more detailed analysis of the results (Section 7.4), and an updated discussion of future work based on recent research developments (Section 9).

2. Related work

2.1. Navigation meshes

Snook [6] and Tozour [7] were among the first to use the term 'navigation mesh' for a subdivision of the walkable space into polygonal regions. Because constructing a navigation mesh by hand is time-consuming and subject to human error, there has been increasing interest in automatically computing a navigation mesh from an input environment.

Voxel-based methods [8–11] usually take an unprocessed 3D environment as their input. To construct a navigation mesh, they discretize the environment into a 3D grid of *voxels* using GPU techniques, extract the voxels that correspond to walkable regions, and summarize this information in a navigation mesh that *approximates* the geometry of $\mathcal{E}_{free}$. This reconstruction is based on the assumption that the environment has a single direction of gravity $\vec{g}$, and that characters are cylinders with a fixed height and (sometimes) a fixed radius. Voxel-based methods can handle arbitrary 3D geometry; the approximation automatically resolves issues caused by e.g. intersecting obstacles. However, the precision and efficiency of these methods depends to a certain degree on the grid resolution. The quality of the navigation mesh depends on how well $\mathcal{E}_{free}$ is extracted from the 3D geometry.

Exact methods [7,12–16] require that the exact geometry of $\mathcal{E}_{free}$ is already known, and that this free space has been pre-processed into one or more planar layers. In exchange, they represent their input *precisely*, and they often have provable worst-case construction times and storage sizes, which implies better scalability to large environments. However, extracting $\mathcal{E}_{free}$ from a 3D environment without using approximations is still a topic of ongoing research [17–19].

Researchers have also investigated navigation meshes for other types of geometry or movement. An environment can be subdivided into 3D volumes to encode height differences and variable vertical clearance [20,21]. Alternatively, one could perform crowd simulations on arbitrary surfaces with no consistent direction of gravity [22,23]. Other methods allow characters to jump between surfaces by either checking for jumping possibilities on the fly [24] or annotating a navigation mesh with jump links beforehand [25,26]. Another possible extension is to consider different 'terrain types' in the environment (such as sidewalks, roads, and grass), where each terrain type has a weight that indicates its attractiveness for navigation. These so-called *weighted* or *heterogeneous* environments require their own kinds of data structures and path-planning algorithms [27,28]. These extensions are outside the scope of our study.

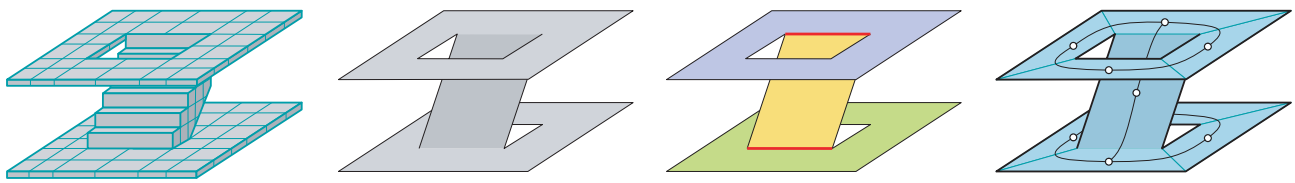
Navigation meshes are useful for simulating crowds of characters with individual properties and goals. Crowd simulation is a large research field with many components including path planning, collision avoidance between characters, animation, and the evaluation of realism. Several books exist that give good overviews of this field [29–32]. Also, there are multiple crowd-simulation frameworks in which navigation meshes play a central role [1,33]. In this paper, we focus on the fundamental properties of navigation meshes, so we will not treat the field of crowd simulation in more detail. On a related note, this paper does not look into path planning itself, but only at the geometric data structures *used* for path planning. However, path planning and crowd simulation are important motivations for many of the properties and metrics that we will propose.

2.2. Comparative studies

Our comparative study is inspired by comparisons for *other* aspects of path planning and crowd simulation. Sturtevant [34] has developed a test set of environments for 2D *grid-based path planning*. This set includes mazes of various sizes and levels from computer games, and it is often used to analyze variants of the A* search algorithm [35]. The same research group has recently launched a similar benchmark set for 3D (voxel) grids [36]. Although we study general navigation meshes rather than grids, we will also include grid-based environments in our experiments.

SteerBench [37] focuses on local behavior such as collision avoidance. It presents a comprehensive set of scenarios that local methods are expected to solve, such as two characters crossing paths, or characters switching places in a narrow corridor. Given the output of a crowd simulation for such a scenario, SteerBench can compute metrics such as the distance that all characters traverse and the amount of energy that they spend. However, the results need to be put in perspective because steering methods typically have many parameters and implementation choices.

Another inspiring example of benchmarking can be found in the field of 3D character animation. Reitsma and Pollard [38] have pre-



(a) 3D environment (b) Walkable environment (c) Multi-layered environment (d) Navigation mesh

Fig. 1. Different representations of an environment, and an example of its navigation mesh. (a) A 3D environment consists of unprocessed 3D geometry. (b) A walkable environment (WE) contains only walkable surfaces. It is the free space $\mathcal{E}_{free}$ of a 3D environment. (c) A multi-layered environment (MLE) is a WE subdivided into layers such that each layer is non-overlapping when projected to 2D. (d) A navigation mesh is a description of a WE for path planning purposes.

sented metrics for objectively evaluating the quality of a motion graph, a data structure used for transitioning between animation clips of (for example) a walking character.

In this paper, we apply the same philosophy to a different field of study. We compare navigation meshes based on metrics, input environments, and a single test platform.

3. Definitions

In this section, we give definitions of environments and navigation meshes. This is useful because all existing papers and algorithms use slightly different terminology; the content of Sections 4 and 6 requires unified definitions.

3.1. 2D environment

A *2D environment* (2DE) is a finite subset of the 2D plane with polygonal holes; we refer to these holes as obstacles. We will not consider point or line segment obstacles in this paper. The obstacle space $\mathcal{E}_{obs}$ is the union of all obstacles. Its complement is the free space $\mathcal{E}_{free}$. Let n be the number of vertices required to define $\mathcal{E}_{obs}$ or $\mathcal{E}_{free}$ using simple polygons. We call n the *complexity* of $\mathcal{E}$.

In our experiments, we want to treat 2D and 3D environments similarly. We will therefore embed our 2DEs in $\mathbb{R}^3$ by assigning a height component of zero to each vertex.

3.2. 3D environment

In this paper, a *3D environment* (3DE) is a raw collection of planar polygons in $\mathbb{R}^3$. These polygons may include floors, ceilings, walls, or any other type of geometry. Fig. 1a shows an example. To define the *free space* $\mathcal{E}_{free}$ of a 3DE, we need parameters that describe on which surfaces a character may walk. Examples of such parameters are the maximum slope with respect to the direction of gravity, the maximum height difference between nearby polygons (e.g. the maximum step height of a staircase), and the required minimum vertical distance between a floor and a ceiling.

Characters are typically approximated by cylinders. Some navigation mesh construction algorithms use a predefined character radius to determine $\mathcal{E}_{free}$, while others do not. In this paper, for any navigation mesh that is based on a predefined radius, we will use a radius of zero to enable an objective comparison to other methods.

3.3. Walkable environment

A *walkable environment* (WE) is a set of interior-disjoint polygons in $\mathbb{R}^3$ on which characters can stand and walk. Thus, a WE is a clean representation of the free space $\mathcal{E}_{free}$ of a 3DE, based on the filtering parameters and character properties mentioned earlier. Any two polygons are directly connected if and only if characters can walk directly between them. Furthermore, polygons in the 3DE that are nearly adjacent are typically merged in the WE; for example, staircases are converted to ramps. Fig. 1b shows an example. In our experiments, all environments will be WEs, so we know beforehand which area is supposed to be covered by a navigation mesh. This 'ground truth' is required for some of the metrics in Section 6.

All polygons in the WE have a maximum slope with respect to the *ground plane* P, which is the plane perpendicular to the gravity direction $\vec{g}$. It is common for a navigation mesh to project the length of a path onto P as well, i.e. to ignore height differences along a path during planning. Therefore, in this paper, we will not judge a navigation mesh by its preservation of height differences.

The complexity of a WE is the total number of polygon vertices, which we denote by n just like in 2D environments. The free space $\mathcal{E}_{free}$ is simply the set of polygons itself. Its complement, the obstacle space $\mathcal{E}_{obs}$, can be thought of as 'anything beyond the boundary of $\mathcal{E}_{free}$', but (unlike in 2D) it is difficult to represent or visualize because it does not necessarily consist of polygons on a plane.

It is important to see that a WE *can be self-overlapping* when projected onto the ground plane P, i.e. it is not guaranteed that all surfaces are visible from a single top view. This strongly influences the construction of navigation meshes: an algorithm for 2DEs cannot easily be applied to WEs in general.

3.4. Multi-layered environment

Some navigation meshes require the WE to be subdivided into 2D components. A *multi-layered environment* (MLE) [11,16] is a subdivision of a WE into *layers* such that the walkable polygons of each individual layer are non-overlapping when projected onto P. The layers are connected by line segments. An example of an MLE is shown in Fig. 1c.

Although layers do 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 other purposes as well, including visualization (each layer can be drawn in 2D) and identification (all points in $\mathcal{E}_{free}$ can be uniquely specified using a 2D position and a layer ID).

The complexity of an MLE is given by the number of layers l, the number of connections k, and the number of boundary vertices n in all layers combined. Converting a WE to an MLE with a minimal number of connections is NP-hard [39], but good results can be obtained using heuristics [40]. In our experiments, we will subdivide all WEs into layers to facilitate the construction of navigation meshes.

A 2DE is essentially an MLE with only one layer (or, equivalently, a WE that can be projected onto P without overlap). Section 6 will define quality metrics for WEs in general, so that 2D and 3D input can be treated equally.

3.5. Navigation mesh

Now that we have a definition of the free space $\mathcal{E}_{free}$, we can define a navigation mesh as a tuple $\mathcal{M} = (\mathcal{R}, \mathcal{G})$:

- $\mathcal{R} = \{R_0, R_1, \ldots\}$ is a collection of geometric *regions* in $\mathbb{R}^3$ that represents $\mathcal{E}_{free}$. Each region R_i is *P-simple*, by which we mean that a region cannot intersect itself when projected onto the ground plane P.
- $\mathcal{G} = (V, E)$ is an undirected *graph* that describes how characters can navigate between the regions in $\mathcal{R}$.

Fig. 1 d shows an abstract example of a navigation mesh. For many navigation meshes, $\mathcal{R}$ consists of non-overlapping simple polygons, and $\mathcal{G}$ is simply the dual graph of $\mathcal{R}$, with one vertex per region and one edge per pair of adjacent region sides. However, other possibilities exist. In the Clearance Disk Graph [11], $\mathcal{R}$ consists of overlapping disks, and $\mathcal{G}$ contains an edge wherever two disks overlap. The Explicit Corridor Map [16] is explicitly defined as a graph, and the mesh regions can be derived from its annotations. Still, all meshes have in common that they represent $\mathcal{R}$ and $\mathcal{G}$ in some way.

3.6. Summary

For further reference throughout this paper, we now summarize the definitions from this section.

- A *3D environment* (3DE) is an unprocessed collection of polygons in $\mathbb{R}^3$.
- A *walkable environment* (WE) is a 'clean' representation of the *free space* $\mathcal{E}_{free}$ of a 3DE, i.e. the surfaces on which characters can stand and walk. We refer to its number of distinct polygon vertices n as its *complexity*.
- A *multi-layered environment* (MLE) is a WE subdivided into l layers connected via k line segments, so that each individual layer can be projected onto the horizontal plane without overlap.
- A *2D environment* (2DE) is an MLE with $l = 1$ and $k = 0$, or (equivalently) a WE that can be entirely seen from a single top view.
- Given an input environment, a *navigation mesh* is a set of regions that is supposed to cover the free space $\mathcal{E}_{free}$, plus a graph that describes how to navigate between regions.

4. Properties of navigation meshes

In this section, we propose a set of properties that describe a navigation mesh's data structure, algorithms, and limitations. These properties do not depend on a specific implementation or environment. They can serve as a 'checklist' to simplify choosing an appropriate mesh for a particular application. In Section 5, we will use these properties to compare various navigation meshes.

Region type: The type of regions in $\mathcal{R}$, e.g. triangles, polygons, or disks.

Graph type: A description of the path planning graph $\mathcal{G}$, e.g. 'the dual graph of $\mathcal{R}$' or 'the medial axis of $\mathcal{E}_{free}$'.

Overlap: Whether the regions in $\mathcal{R}$ can overlap by definition. Generally, overlap is discouraged because it may cause problems for geometric algorithms that assume non-overlapping input. Also, it may complicate path planning and crowd simulation if a query point (i.e. an agent) can be in multiple regions at the same time.

Pipeline: The conversion steps performed by the construction algorithm, e.g. "from a 2D environment to a navigation mesh" or "from a 3DE via an MLE to a navigation mesh". We also indicate whether this pipeline is voxel-based or exact.

Parameters: The parameters that the user needs to set for the construction algorithm of the navigation mesh. Having fewer parameters implies a more automated process for computing the mesh. These parameters are often related to the filtering process that extracts the walkable surfaces from the 3D geometry.

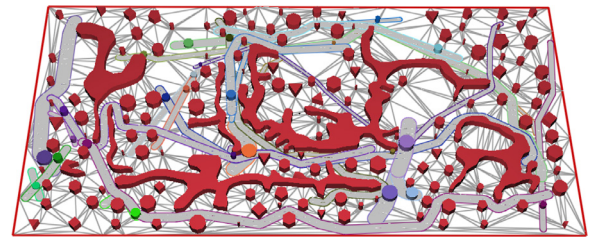


Fig. 2. An example image of the LCT navigation mesh. Image courtesy of Kallmann [14].

Computational complexity: The asymptotic construction time of the navigation mesh. This is usually expressed in terms of the environment complexity or a grid resolution.

Storage complexity: The asymptotic size of the navigation mesh data structure.

Clearance: Whether the navigation mesh supports the computation of paths with an arbitrary clearance from obstacles, i.e. paths for disks with an arbitrary radius.

Dynamic updates: Whether the mesh supports dynamic insertions and deletions of obstacles.

5. Theoretical comparison

Based on the theoretical properties listed in Section 4, we now describe and compare the state-of-the art navigation meshes that will also be included in our experiments in Section 7. The first two navigation meshes are exact; the others are voxel-based and cover the full 3D pipeline. Although more navigation meshes exist, we currently include only the navigation meshes that are designed specifically for the types of environments described in Section 3, and for which we could obtain robust source code from their respective authors. Table 1 summarizes our comparison using the properties from Section 4.

For each navigation mesh, we will include a representative example image obtained from its corresponding publication or from its software. For more examples, we refer the reader to Figs. 11–23, which give side-by-side visual impressions of our experiment results.

5.1. Local Clearance Triangulation

The Local Clearance Triangulation (LCT) by Kallmann [14] (see Fig. 2) subdivides a 2D environment of complexity n into $\mathcal{O}(n)$ triangles with all vertices on the boundary of $\mathcal{E}_{free}$. These triangles are the regions of $\mathcal{R}$, and $\mathcal{G}$ is their dual graph. Each triangle edge is annotated with *clearance* values encoding minimum distances to the relevant obstacles when crossing this edge. Therefore, during path planning, the LCT can determine in constant time whether a particular move is collision-free for a character with a particular radius.

The triangles of the LCT need to adhere to certain local constraints in order for their clearance annotations to work. The LCT is constructed by first computing a constrained Delaunay triangulation in $\mathcal{O}(n \log n)$ time, and then applying $\mathcal{O}(n)$ refinements until all constraints are satisfied. This sequence of refinements may take $\mathcal{O}(n^2)$ time in the worst case. The tested implementation runs in $\mathcal{O}(n\sqrt{n})$ expected time by using a special point-location method. The LCT also supports dynamic updates. An extension to MLEs has not been developed, but an approach similar to the Explicit Corridor Map (see next subsection) should be possible.

The LCT takes a set of 2D line segment constraints as input, so our benchmark program first needs to compute the boundary

Table 1
Overview of the navigation meshes compared in this paper. In the "Construction time" and "Storage" columns, n indicates the number of distinct polygon vertices of the input environment, k is the number of connections in an MLE, and S is the number of grid cells used by a voxel-based algorithm. '**?**" means that an algorithm and its implementation are so tightly coupled that a reliable asymptotic bound cannot be given. For the rightmost columns, '**+**" means that a property is supported in the current implementation, '**+/-**' means that a property is currently not supported but can be added, and '**-**' means that a property is not supported by definition.

Navigation mesh	Region type	Graph type	Overlap	Pipeline	Parameters	Construction time	Storage	Dynamic updates	Arbitrary clearance
LCT	Triangles	Dual of $\mathcal{R}$	No	2D $\rightarrow$ $\mathcal{M}$ (exact)	None	2D: $\mathcal{O}(n\sqrt{n})$ (expected)	$\mathcal{O}(n)$	+	+
ECM	Polygons	Medial axis	No	2D/MLE $\rightarrow$ $\mathcal{M}$ (exact)	None	2D: $\mathcal{O}(n \log n)$ MLE: $\mathcal{O}(kn \log n)$	$\mathcal{O}(n)$ $\mathcal{O}(kn)$	+	+
CDG	Disks	Dual of $\mathcal{R}$	Yes	3D $\rightarrow$ $\mathcal{M}$ (voxel-based)	3D filtering Voxel precision Min character radius Min/max disk size	?	$\mathcal{O}(S)$	+/-	+
Recast	Convex polygons	Dual of $\mathcal{R}$	No	3D $\rightarrow$ $\mathcal{M}$ (voxel-based)	3D filtering Voxel precision Region refinement Character radius	?	?	+/-	-
NEOGEN	(Convex) polygons	Dual of $\mathcal{R}$	No	3D $\rightarrow$ MLE $\rightarrow$ $\mathcal{M}$ (voxel-based + exact)	3D filtering Voxel precision Convexity relaxation	2D: $\mathcal{O}(n^2)$ MLE: $\mathcal{O}(n^2)$ 3D: ?	$\mathcal{O}(n)$ $\mathcal{O}(n)$	+/-	+/-
Grid	Squares	Dual of $\mathcal{R}$	No	3D $\rightarrow$ $\mathcal{M}$ (voxel-based)	3D filtering Voxel precision	?	$\mathcal{O}(S)$	+	-

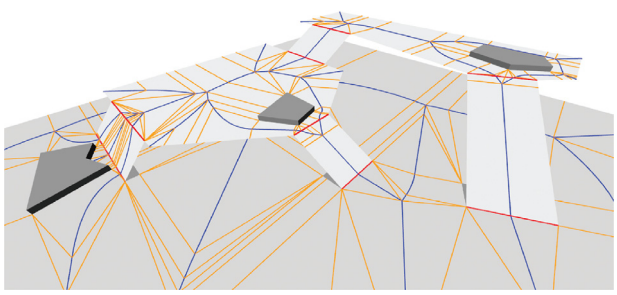


Fig. 3. An example image of the ECM navigation mesh. Image courtesy of van Toll et al.[41].

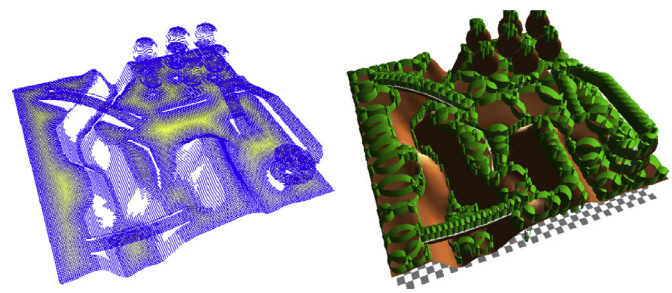


Fig. 4. An example image of the CDG, showing walkable voxels (left) and the resulting navigation mesh (right). Image courtesy of Pettré et al.[11].

segments of an environment. The output of the LCT program is a set of constrained and unconstrained segments, and the set of triangles conforming to these segments. In our comparison, we will filter out the triangles that lie inside the obstacle space. These preprocessing and post-processing steps will not be included in our time measurements.

5.2. Explicit Corridor Map

The Explicit Corridor Map (ECM) by Geraerts et al.[12,41] (see Fig. 3) is an exact navigation mesh. Its graph $\mathcal{G} = (V, E)$ is the medial axis of $\mathcal{E}_{free}$, where V contains the medial axis vertices of degree 1, 3, or higher. Each edge in E is a sequence of medial axis arcs between two vertices of V. An edge consists of its two endpoints and a sequence of bending points at which the medial axis changes shape. These points are all annotated with their nearest obstacle point on the left and right side of the medial axis. This induces a subdivision of $\mathcal{E}_{free}$ into polygonal regions. Each region in $\mathcal{R}$ is a (possibly non-convex) polygon of at most 6 distinct vertices.

Because the distance to the nearest obstacle is known at each bending point, the ECM can be used to plan paths for characters of an arbitrary radius. The nearest obstacle point for any query point q can be found in constant time when the region containing q is known (which is not the case for polygon subdivisions in general). The ECM also supports dynamic updates.

For a 2D environment of complexity n, the ECM has size $\mathcal{O}(n)$ and is computed in $\mathcal{O}(n \log n)$ time. For an MLE with k connections as defined in Section 3, the *medial axis* has size $\mathcal{O}(n)$ and can be computed in $\mathcal{O}(n \log n \log k)$ time by iteratively opening the connections. The implementation used for this paper runs in $\mathcal{O}(kn \log n)$ time, and it splits the ECM's edges whenever they intersect a connection, which yields a size of $\mathcal{O}(kn)$. In exchange for its advantages, the ECM is conceptually slightly more difficult than e.g. a triangulation; it may be a less intuitive choice at first sight.

5.3. Clearance Disk Graph

Pettré et al.[11] presented the first *voxel-based* navigation mesh for 3D environments (see Fig. 4). They introduced many new concepts that have evolved in later algorithms. This navigation mesh does not have an official name; in this paper, we refer to it as the Clearance Disk Graph (CDG).

The CDG uses voxelization to approximate where characters can stand. Next, the voxels are extracted for which the clearance (the distance to the nearest obstacle) is locally largest. These form an approximation of the medial axis of $\mathcal{E}_{free}$. Each remaining voxel has an obstacle-free disk associated to it. A subset of these voxels is chosen such that their disks overlap but do not contain each other's centers. The resulting disks are the regions of $\mathcal{R}$, and the graph $\mathcal{G}$ has a vertex for each disk and an edge for each pair of overlapping disks. A disadvantage of the CDG is that its disks can never fully cover the free space. Extra disks (that do not lie on the medial axis) can be added to improve coverage.

The asymptotic construction time of the CDG is difficult to assess because the algorithm relies on rendering techniques. Also, the number of disks cannot be expressed in terms of the environment complexity, but it is at least limited by the number of voxels S.

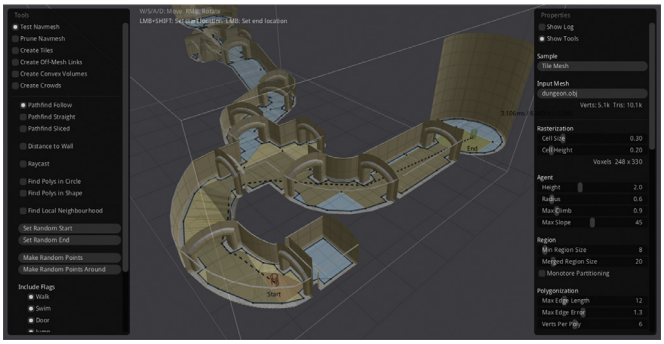


Fig. 5. An example image of the Recast navigation mesh. This is a screenshot of the Recast software by Mononen [9].

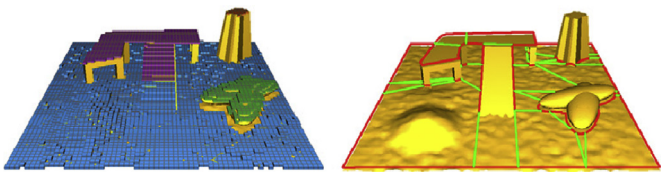


Fig. 6. An example image of NEOGEN, showing walkable voxels (left) and the resulting navigation mesh (right). Image courtesy of Oliva and Pelechano [10].

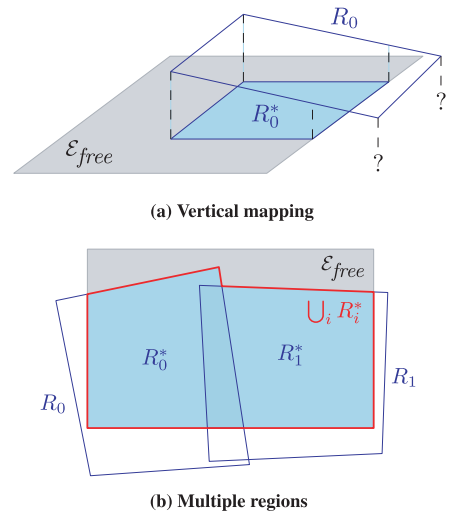


Fig. 7. Mapping and coverage. (a) 3D view of a navigation mesh region R_0. Only a part of R_0 can be vertically mapped onto $\mathcal{E}_{free}$. The mapped region R_0^* is highlighted in blue. (b) Top-view of a different example with two regions R_0 and R_1. The mapped regions R_0^* and R_1^*, highlighted in blue, partly overlap. The union of all mapped regions, $\bigcup_i R_i^*$, is well-defined because the mapped regions are subsets of $\mathcal{E}_{free}$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5.4. Recast

The Recast Navigation toolkit by Mononen [9] (see Fig. 5) also uses voxelization to approximate $\mathcal{E}_{free}$. However, unlike the CDG, Recast converts the walkable voxels to non-overlapping convex *polygonal* regions. The method offers many detailed settings for this conversion, e.g. for tracing obstacle boundaries, grouping adjacent polygons, and discarding regions that are too small. This large number of parameters complicates a comparative study because each environment may have its own optimal settings. Another parameter is the character radius, which is subtracted from the navigation mesh during its construction. As mentioned, we will use a radius of zero to allow a fair comparison to other methods.

Recast computes two versions of the navigation mesh: a coarse mesh that is used for path planning, and a detailed mesh that captures height differences more accurately. In our experiments, we will use the coarse mesh to determine the regions $\mathcal{R}$ and the graph $\mathcal{G}$, and the detailed mesh to measure how well the result covers $\mathcal{E}_{free}$.

Recast does not provide theoretical guarantees of running times, accuracy, or storage size. However, the source code of Recast is considered to be fast, robust, and a popular choice for game development. A variant of Recast is included in the popular Unity3D game engine [42].

5.5. NEOGEN

The NEOGEN method by Oliva and Pelechano [10] (see Fig. 6) also starts with voxelization, but it then groups the walkable voxels into 2D layers, which yields an approximation of an MLE. For each layer, NEOGEN then uses GPU techniques to obtain a more precise floorplan in a way that does not depend on the voxel size. Compared to Recast, the overall precision of NEOGEN is therefore less dependent on the grid resolution.

Based on these floorplans, an exact 2D algorithm [15] is used to compute the navigation mesh for each layer. This 2D algorithm subdivides the free space of a layer into convex polygons. For each convex obstacle corner, the algorithm draws line segments to other obstacles within an angular range. This algorithm runs in $\mathcal{O}(nr)$ time, where n is the total number of obstacle vertices, and $r \in \mathcal{O}(n)$ is the number of convex obstacle corners. Finally, the navigation meshes of each layer are merged into a single data structure. In our experiments, for simplicity, we will use the "full" voxel-based method in both MLEs and 2D environments.

A contribution of NEOGEN is the *convexity relaxation* parameter that can be used to allow slightly non-convex regions. This decreases the total number of regions in exchange for having more complex region shapes. NEOGEN does not use a predefined character radius, and its regions do not encode clearance information by default. However, such information could be added if desired [43].

5.6. Grid

As a baseline for our comparison, we have implemented a simple grid method. It voxelizes the environment similarly to Recast and NEOGEN, but it uses the walkable voxels directly as navigation mesh regions. Therefore, each region in $\mathcal{R}$ is a square.

We include this method because grids are still frequently used for path planning. They are easy to implement and a common choice for games that are grid-based by design [34]. Another advantage is that algorithms such as A* search can be optimized for grids [44–47]. Many variants of A* are either designed with grids in mind [48] or explained in terms of grids [49–51]. Another advantage of grids is that they are relatively easy to update dynamically when obstacles appear or disappear, because a grid does not have to satisfy advanced geometric properties such as in triangulations or Voronoi diagrams.

However, grids are typically more dense than other navigation meshes (e.g. they require many cells to represent large open spaces), which makes them less suitable for planning many paths in real-time. Also, a high grid resolution is required if the environment contains many details, and a grid cannot fully cover the free space if the input geometry is not axis-aligned. To alleviate resolution-related issues, there are several ways to use a dynamic grid-cell size [52,53]. Such adaptive variants of 2D or 3D grids are not included in our study for now.

5.7. Comparison

The distinction between exact and voxel-based navigation meshes is clear. Exact methods have the advantages of provable running times, scalability to large environments, and guaranteed precision. However, if an application features 3D geometry that has not yet been pre-processed into planar layers, then these exact methods cannot be applied immediately. By contrast, voxel-based methods offer a full conversion pipeline from raw 3D geometry to a navigation mesh. These methods are less scalable, and we expect that they cannot achieve perfect coverage when an environment is very detailed or not axis-aligned.

The LCT and ECM have many properties in common: they are exact, they do not require any parameters, they encode clearance information, and their size is $\mathcal{O}(n)$. A difference is that the ECM currently supports MLEs as well. Also, the ECM construction algorithm has a better asymptotic worst-case running time, but this difference might only be noticeable at a very high complexity or in very uncommon types of environments. Therefore, we expect that the two algorithms will perform similarly in practice.

NEOGEN and Recast both convert voxels to convex polygonal regions. Recast has more parameters: it aims at a semi-automatic construction process in which the user tweaks parameters to achieve the desired result. The main advantages of NEOGEN are its techniques to obtain a higher precision per layer, the use of an exact 2D algorithm, and the convexity relaxation parameter. An advantage of Recast is the maturity of its code.

The CDG has similar parameters to NEOGEN for filtering the 3D environment, but its representation with overlapping disks is different. In polygonal environments, a finite set of disks cannot cover the free space completely. However, the advantage of disks is that they trivially encode clearance information. Furthermore, the algorithm for computing a CDG is relatively easy to understand and implement. A practical matter to take into account is that the CDG source code is not optimized for e.g. gaming applications; we therefore expect that the other methods will be more efficient in their current state.

Finally, we expect that our naive grid implementation yields the largest graphs, and that it does not cover $\mathcal{E}_{free}$ well if the obstacles are not aligned with the grid cells. Other voxel-based methods use post-processing steps to convert voxels to smooth regions, which improves coverage and simplifies the graph.

6. Quality metrics for navigation meshes

For a navigation mesh $\mathcal{M} = (\mathcal{R}, \mathcal{G})$ that has been constructed for an environment using a particular implementation, we want to objectively measure the quality. Many possible evaluation criteria exist, and each application area may have its own view of what is good or desirable. In this paper, we choose to focus on the navigation mesh itself and on the performance of its construction algorithm. We will present metrics that answer the following questions:

- **Coverage.** How accurately do the regions of $\mathcal{R}$ cover the geometry of $\mathcal{E}_{free}$? If parts of the free space are not covered, characters might not find a path in $\mathcal{G}$ even though a path exists in $\mathcal{E}_{free}$. If parts outside the free space are covered, characters might accidentally find paths through obstacles.
- **Connectivity.** How accurately does the graph $\mathcal{G}$ represent the connectivity of $\mathcal{E}_{free}$? This question is related to coverage because it determines whether or not appropriate paths can be found; however, it concerns topology rather than geometry.
- **Complexity.** How efficiently does $\mathcal{M}$ represent $\mathcal{E}_{free}$, i.e. how 'compact' is the mesh? This can refer to the size of the graph $\mathcal{G}$ (a smaller graph allows faster path planning queries) or to the complexity of each individual region in $\mathcal{R}$ (simpler regions allow faster basic operations such as point location). It depends on the application which property is more desirable.
- **Performance.** How efficiently is $\mathcal{M}$ computed in terms of time and memory? An efficient algorithm allows the construction of navigation meshes in interactive applications such as level editors. Even if the navigation mesh is precomputed in an off-line stage, performance is still desirable.

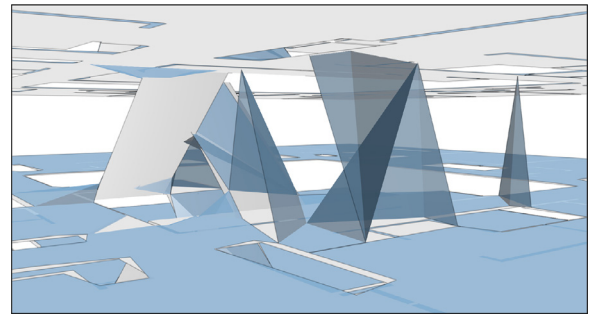


Fig. 8. Partial rendering of the Recast navigation mesh for the *Tower* environment. The ground truth is shown in gray. The navigation mesh (shown in blue) connects surfaces incorrectly, which is reflected in bad connectivity values. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Of course, many other interesting questions could be asked, e.g. related to the peformance of path planning queries, or to the quality or realism of paths. We will discuss several options in Section 9 as suggestions for future work.

Analyzing coverage and connectivity is only useful for voxel-based navigation meshes that attempt to 'discover' $\mathcal{E}_{free}$ themselves; exact methods are known to yield perfect coverage. Also, some of our questions can only be answered if the ground truth (the structure of $\mathcal{E}_{free}$) is known. Therefore, each input environment in our experiments will be a 'clean' walkable environment, i.e. a manifold that contains only walkable polygons. While this implies that voxel-based methods will not fully use their advantage of handling raw (non-clean) 3D geometry, we believe that using the same input for all methods yields a more objective comparison.

Since the outcome of each metric depends on implementation details, the results should always be judged in combination with the theoretical properties of Section 4.

6.1. Coverage

The first set of metrics describes how well the free space is covered. Coverage is a complicated property to evaluate due to the 3D structure of $\mathcal{R}$ and $\mathcal{E}_{free}$. We need to introduce a number of concepts before we can define actual metrics. These concepts assume that the environment has a consistent direction of gravity. Coverage is the only category of metrics in which this assumption comes into play.

6.1.1. Mapping the navigation mesh onto the free space

Comparing the geometry of $\mathcal{R}$ to the geometry of $\mathcal{E}_{free}$ requires us to vertically map these two structures onto each other. This is straight-forward if the environment consists of a single layer because everything can then be projected onto the ground plane P. However, for general WEs in $\mathbb{R}^3$, mapping $\mathcal{R}$ onto $\mathcal{E}_{free}$ is ambiguous. In an abstract sense, we require a function m such that, for any point p in a navigation mesh, $m(p)$ *vertically* maps p to an appropriate point in $\mathcal{E}_{free}$ if possible. If this mapping is not possible, p is assumed to lie outside $\mathcal{E}_{free}$. Several choices can be made in defining m precisely, such as the maximum allowed height differ-

ence between p and $m(p)$. We will describe our own implementation of m in Section 7.

Using the function m, we define a *mapped region* R_i^* as a region $R_i$that has been mapped onto $\mathcal{E}_{free}$ wherever possible, i.e. $R_i^* = \{m(p) \mid p \in R_i \text{ and } m(p) \text{ exists}\}$.

Fig. 7 a shows an example of a region and its mapping. Because each mapped region is a subset of $\mathcal{E}_{free}$, we can use the mapped regions to define unions, coverage, and overlap. This allows us to properly compare the navigation mesh to the free space. An example is illustrated in Fig. 7b.

Let the *mapped region set* $\mathcal{R}^*$ be a version of $\mathcal{R}$ in which all regions have been mapped onto $\mathcal{E}_{free}$, i.e. $\mathcal{R}^* = \{R_i^* \mid R_i \in \mathcal{R}\}$. The regions in $\mathcal{R}^*$ may overlap: in that case, some points of $\mathcal{E}_{free}$ are represented more than once.

6.1.2. Computing the projected area

Because we ignore height differences in our problem domain, our coverage metrics are based on projected areas onto the ground plane P. We define the *projected area* of a shape S as follows:

- If S does not overlap itself when projected onto P (i.e. if S is a P-simple shape as defined in Section 3.5), the projected area $||S||$ is the signed area of the projection of S onto P.
- Otherwise, let $\{S_0, \ldots, S_{s-1}\}$ be any subdivision of S into P-simple shapes. The projected area of S is the sum of projected areas of these components, i.e. $||S|| = \sum_i ||S_i||$.

We assume that $\mathcal{E}_{free}$ is given as a subdivision into P-simple shapes, such that $||\mathcal{E}_{free}||$ can be computed.

6.1.3. Coverage metrics

We introduce three coverage metrics. Each metric has a regular version M with range $\mathbb{R}_{\geq 0}$ (measured in m^2), and a normalized version M' with range [0, 1], as described below.

Free space covered: The area of $\mathcal{E}_{free}$ that is correctly covered by at least one navigation mesh region. Because the regions in $\mathcal{R}^*$ may overlap and we do not want to count overlapping regions twice, we first compute the union of $\mathcal{R}^*$ in $\mathbb{R}^3$. It is desirable that a navigation mesh has high coverage: this allows characters to use more of $\mathcal{E}_{free}$.

$$Cov = ||\bigcup_i R_i^*|| \quad \text{and} \quad Cov' = \frac{Cov}{||\mathcal{E}_{free}||}$$

Incorrect area: The area of the mesh that could not be mapped to $\mathcal{E}_{free}$, i.e. the area of the mesh that 'overshoots' $\mathcal{E}_{free}$ and lies in the obstacle space. Intuitively, this is the difference between $\mathcal{R}$ and the part of $\mathcal{R}$ that can be mapped onto $\mathcal{E}_{free}$. Ideally, the incorrect area should be zero because areas outside $\mathcal{E}_{free}$ should not be accessible to characters.

$$A_{inc} = \sum_i (||R_{i||} - ||R_i^*||) \quad \text{and} \quad A'_{inc} = \frac{A_{inc}}{\sum_i ||R_i||}$$

Note: while it may seem more intuitive to express this metric as 'the area of the obstacle space $\mathcal{E}_{obs}$ that is covered', this would be impossible because $\mathcal{E}_{obs}$ does not have an area for WEs in 3D.

Overlap: The amount of overlap among the regions in the navigation mesh. Intuitively, overlap is the sum of all region areas minus the area that is covered at least once. Because coverage is only defined properly inside $\mathcal{E}_{free}$, overlap is also based on the mapped region set $\mathcal{R}^*$. The normalized version indicates which fraction of $\mathcal{R}^*$ is redundant.

$$Ov = \sum_i ||R_i^*|| - ||\bigcup_i R_i^*|| \quad \text{and} \quad Ov' = \frac{Ov}{\sum i ||R_i^*||}$$

If a navigation mesh is deliberately based on overlapping regions (such as the Clearance Disk Graph [11]), then this metric simply indicates how much space is covered more than once. Otherwise, overlap most likely indicates a bug in the navigation-mesh implementation, and this metric helps detect problems without requiring tedious visual inspection.

6.2. Connectivity

The second set of metrics analyzes how well the graph $\mathcal{G} = (V, E)$ represents the dual graph of $\mathcal{E}_{free}$. These metrics concern topology whereas coverage metrics are related to geometry.

 # Connected components: The number of connected components in $\mathcal{G}$. Ideally, this value is equal to the number of connected components in $\mathcal{E}_{free}$. Having more components implies that not all adjacencies in $\mathcal{E}_{free}$ are captured. Having fewer components implies that regions have been made adjacent while there are actually obstacles in-between.

 # Boundaries: The number of environment boundaries perceived by the navigation mesh. Ideally, this value is equal to the actual number of boundaries of $\mathcal{E}_{free}$. It can be computed by traversing the graph $\mathcal{G}$, checking the corresponding regions in $\mathcal{R}$, and collecting the region edges that are not shared by multiple regions. The number of boundaries is the number of closed loops that are traced. Note: if the number of boundaries perfectly matches that of $\mathcal{E}_{free}$, this does not automatically mean that the *geometry* of $\mathcal{R}$ is correct. This is one example of the fact that the metrics of different categories should be interpreted in a combined manner.

6.3. Complexity

The third set of metrics measures how efficiently the navigation mesh represents $\mathcal{E}_{free}$. The size of $\mathcal{G}$, the number of regions in $\mathcal{R}$, and the complexity of these regions may affect the efficiency of path planning and crowd simulation.

 # Vertices, # Edges: The number of vertices and the number of edges in $\mathcal{G}$, i.e. $|V|$ and $|E|$. A larger graph implies that path planning queries (and other algorithms that browse the graph) typically take more time to answer. Therefore, lower numbers imply faster path planning.

 # Regions: The number of regions in the navigation mesh: $|\mathcal{R}|$. This indicates how efficiently the free space is represented by elementary parts. It also suggests how often a character in a (crowd) simulation may move from one region to another. Moving to another region typically triggers computational overhead in such a simulation. Hence, having fewer regions may cause some aspects of the simulation to run more efficiently.

If $\mathcal{G}$ is simply the dual graph of $\mathcal{R}$, then $|\mathcal{R}| = |V|$. In our study, the ECM is the only navigation mesh for which $|\mathcal{R}|$ and $|V|$ may be different.

Region complexity: The number of floating-point numbers required to describe the regions in $\mathcal{R}$. Since we treat regions as shapes in $\mathbb{R}^3$, we will say that a polygonal region with p vertices has complexity $3p$, and that a disk has a complexity of 4 (a center point in $\mathbb{R}^3$ and a radius). Some navigation meshes have extra annotations, such as the maximum allowed radius of a character for an edge traversal [14]. We do not include such annotations in this metric. We measure the average complexity among all regions, the standard deviation, and the total complexity of all regions combined. A low region complexity implies that geometric operations within these regions are computationally cheap.

6.4. Performance

The final set of metrics concerns the practical performance of a navigation mesh implementation. One issue to take into account is that voxel-based methods include a conversion from a 3DE to

Table 2
Summary of the navigation mesh quality metrics described in Section 6.

Metric	Range	Preferred value
Free space covered (m^2): Cov	$\mathbb{R}_{\geq 0}$	Ground truth
Normalized: Cov'	[0,1]	1
Incorrect area (m^2): A_{inc}	$\mathbb{R}_{\geq 0}$	0
Normalized: A'_{inc}	[0, 1]	0
Overlap (m^2): Ov	$\mathbb{R}_{\geq 0}$	0
Normalized: Ov'	[0, 1]	0
# Connected components	$\mathbb{N}$	Ground truth
# Boundaries	$\mathbb{N}$	Ground truth
# Graph vertices: $\|V\|$	$\mathbb{N}$	As small as possible
# Graph edges: $\|E\|$	$\mathbb{N}$	As small as possible
# Regions: $\|R\|$	$\mathbb{N}$	As small as possible
Region complexity	Total: $\mathbb{N}$ Avg/SD: $\mathbb{R}_{\geq 0}$	As small as possible
Construction time (ms)	Avg/SD: $\mathbb{R}_{\geq 0}$	As small as possible
Memory usage (MB)	Avg/SD: $\mathbb{R}_{\geq 0}$	As small as possible

a WE, whereas exact methods do not. Another issue is that different implementations are in drastically different states: some are a 'proof of concept' for research purposes, while others are highly optimized for the gaming and simulation industry. Still, these metrics can indicate if an implementation corresponds to the asymptotic complexity of a navigation mesh, and how well a navigation mesh scales to large or complex environments.

Construction time: The time (in milliseconds) spent on computing the navigation mesh. Naturally, fast construction is encouraged because it makes the algorithm suitable for interactive applications.

Memory usage: The maximum amount of memory (in MB) used during the execution of the program. A small value implies that the mesh can be computed in situations with limited resources, e.g. on a game console with little working memory.

To obtain more reliable results in this category, we will run each navigation mesh program 10 times and report the average values and standard deviations. This is not needed for the other categories of metrics because the *output* of each program is deterministic.

6.5. Summary

Table 2 summarizes all metrics described in this section. The metrics for coverage and connectivity are easy to interpret because we know their optimal values. The metrics for performance are also intuitive: the more efficiently a mesh is constructed, the better. The complexity metrics are more difficult to judge because not all values can be minimized at the same time: for example, a small set of regions will typically imply that the regions themselves are complex.

Different applications may assign different priorities to each metric. For instance, in games where the mesh needs to be computed at run-time, it is likely that efficiency and real-time performance are preferred over exact coverage. By providing all metrics, we leave their interpretation to the developers of the application at hand. We emphasize that choosing a navigation mesh will always imply a trade-off of the advantages and disadvantages of each algorithm. Not all metrics can be optimized simultaneously, and the theoretical properties from Section 4 are at least equally important.

7. Experimental comparison

In this section, we use our metrics to experimentally compare various navigation meshes in a range of environments. All experiments were run on a Windows 10 laptop with a 3.10 GHz Intel i7-7920HQ CPU, an NVIDIA Quadro M2200 GPU, and 32 GB of RAM.

The supplementary files of this paper contain all input environments (described in Section 7.3) and the output of all navigation meshes. Next to providing OBJ files for all input and output (which can be opened in any 3D viewer or editor), we also provide output in two easy-to-read custom file formats: ADJ files that describe graphs, and MSH files that combine the graph and the regions into one file.

7.1. Implementation

We have converted each navigation mesh program to a standalone executable that reads an input file, computes a navigation mesh, and returns the result. We have written a benchmark tool that communicates with these programs, converts environments between different file formats, and calculates all metrics.

The CDG and the grid method require the walkable surfaces to be visible from all sides. For these two methods, we extrude all input polygons downwards by a small amount.

An important implementation detail is our choice of the mapping function m that is used to compute coverage. For a point p in the navigation mesh, we define $m(p)$ as the *nearest* point in $\mathcal{E}_{free}$ above or below p up to a threshold distance T. The threshold distance is required to prevent erroneous points of $\mathcal{R}$ from getting mapped onto surfaces that are too far away. We choose a value of $T = 1$ meter because the vertical clearance is at least 2 meters in all our test environments. Admittedly, this choice for m requires that the height coordinates of the navigation mesh are sufficiently close to the ground truth. It may fail in environments with gradual yet large height differences that are not captured by the navigation mesh. (In 2D environments, T can be ignored because a vertical mapping is already unambiguous.)

To compute coverage for the CDG navigation meshes, we approximated the disks of the CDG by polygons of 16 vertices. We used *inner* approximations: the approximated disks were smaller than the actual disks. This leads to slightly lower numbers for coverage, incorrect area, and overlap, but the chosen precision is sufficient for comparative purposes.

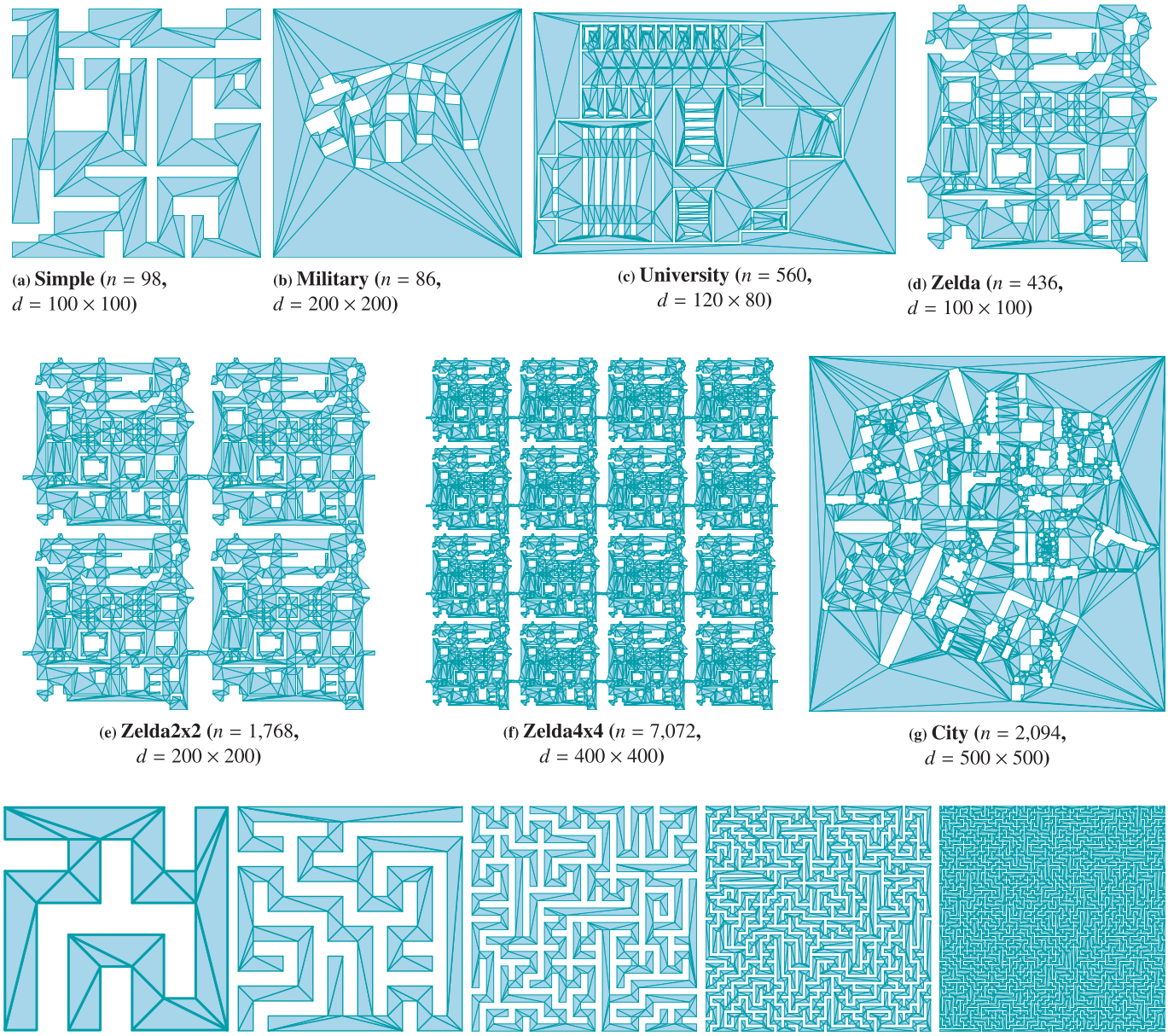
7.2. Parameter settings

Most navigation meshes are built based on various parameters. For simplicity, we use one set of parameter settings for all experiments. These settings are described below.

Precision. For the CDG, Recast, and NEOGEN, we used voxels of $0.1 \times 0.1 \times 0.2$ m. This precision was sufficiently high to generate navigation meshes without large artifacts. (In our previous publication, we used voxels of $0.2 \times 0.2 \times 0.2$ m instead, which caused unfair artifacts in the output of Recast for several 2D environments.) With these standard settings, the *BigCity* environment caused most navigation-mesh programs to crash, and the *City* environment let Recast return empty output. Thus, for these two environments alone, we let Recast use the coarser voxels of $0.2 \times 0.2 \times 0.2$ m in order to obtain a result.

For the CDG, we enforced a maximum resolution of 512 pixels in all dimensions, to keep the construction times manageable for large environments. For the grid, we reduced the voxel size to 1×1 m in the horizontal plane to prevent the program from taking too much time and memory. We could keep the voxel height at 0.2 m. For the CDG, we used a minimum disk radius of 0.1 m (to prevent the method from generating many small disks) and a maximum disk radius of 100, 000 m (which is essentially infinite in our examples).

Filtering. We have created our input environments such that they are entirely walkable and no more surfaces need to be filtered



(a) **Simple** (n = 98, d = 100 × 100)
(b) **Military** (n = 86, d = 200 × 200)
(c) **University** (n = 560, d = 120 × 80)
(d) **Zelda** (n = 436, d = 100 × 100)
(e) **Zelda2x2** (n = 1,768, d = 200 × 200)
(f) **Zelda4x4** (n = 7,072, d = 400 × 400)
(g) **City** (n = 2,094, d = 500 × 500)
(h) **Maze8** (n = 28, d = 7 × 7)
(i) **Maze16** (n = 82, d = 15 × 15)
(j) **Maze32** (n = 356, d = 31 × 31)
(k) **Maze64** (n = 1,390, d = 63 × 63)
(l) **Maze128** (n = 5,574, d = 127 × 127)

Fig. 9. Top views of the 2D environments used in our experiments. The number of polygon vertices n and the physical dimensions d (in meters) are shown in brackets.

out. Therefore, for all voxel-based methods, we use very lenient filtering parameters to ensure that these environments could (in theory) be covered completely. We used a maximum surface slope of 75 degrees, a character radius of 0, and a character height of 0.5 m.

Recast offers various other parameters that would ideally need to be tweaked for each environment to get the best results. We used the following settings: Tiling: Off; Max climb: 0.5; *Min region size: 0; Merged region size: 100;* Partitioning: Watershed; *Max edge length: 100;* Max edge error: 1.3; Vertices per polygon: 6; Detail mesh sample distance: 6; Detail mesh max error: 1. The parameters printed in italics are the only ones who do not have their default value suggested by the Recast software. These are all post-processing parameters that determine how any detected surfaces are merged or removed. Setting 'Min region size' to 0 means that no surfaces get removed for being too small. This reduces the risk of obtaining bad coverage values. Setting 'Merged region size' and 'Max edge length' to a high value means that Recast can merge more regions into one, which reduces the complexity of the navigation mesh. We have experimented with even higher values, but these caused unexpected geometric errors in some of our environments.

Other. To compute ECMs, we used the implementation based on the Boost Voronoi library [54]. It computes a 2D Voronoi diagram in $\mathcal{O}(n \log n)$ worst-case time, and it can process multiple layers of an MLE on parallel threads.

For NEOGEN, we used a convexity relaxation parameter of 0. Increasing this parameter leads to fewer regions (i.e. a smaller graph) and a higher region complexity. Thus, the 'best' value for this parameter depends on the application.

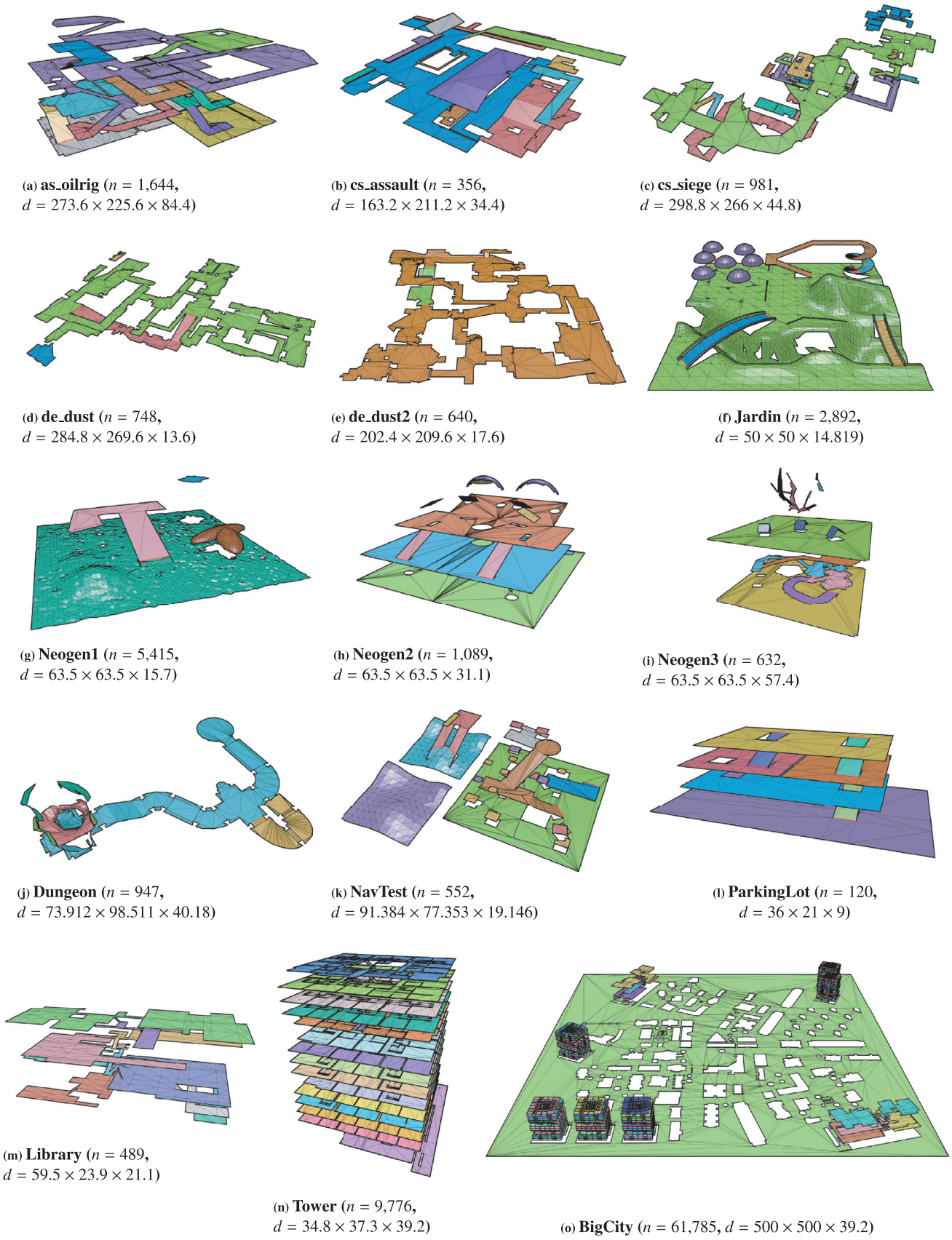


Fig. 10. Renderings of the multi-layered environments used in our experiments. Each layer of an environment is shown in a different color. Connections between layers are shown in red. The number of polygon vertices n and the physical dimensions d (width × depth × height, in meters) are shown in brackets. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

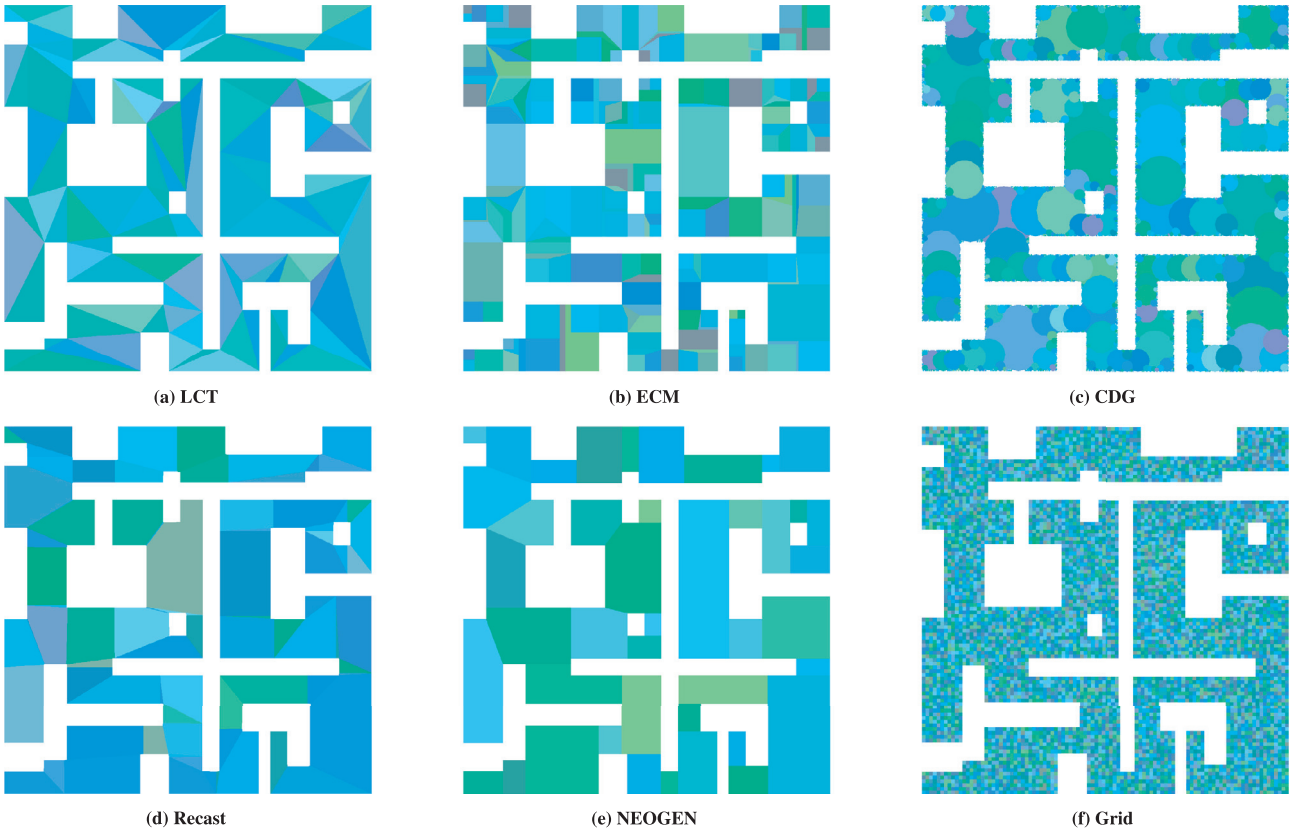


Fig. 11. Navigation meshes computed for the *Simple* environment. Regions are shown in different colors. The corresponding graphs have been omitted for clarity.

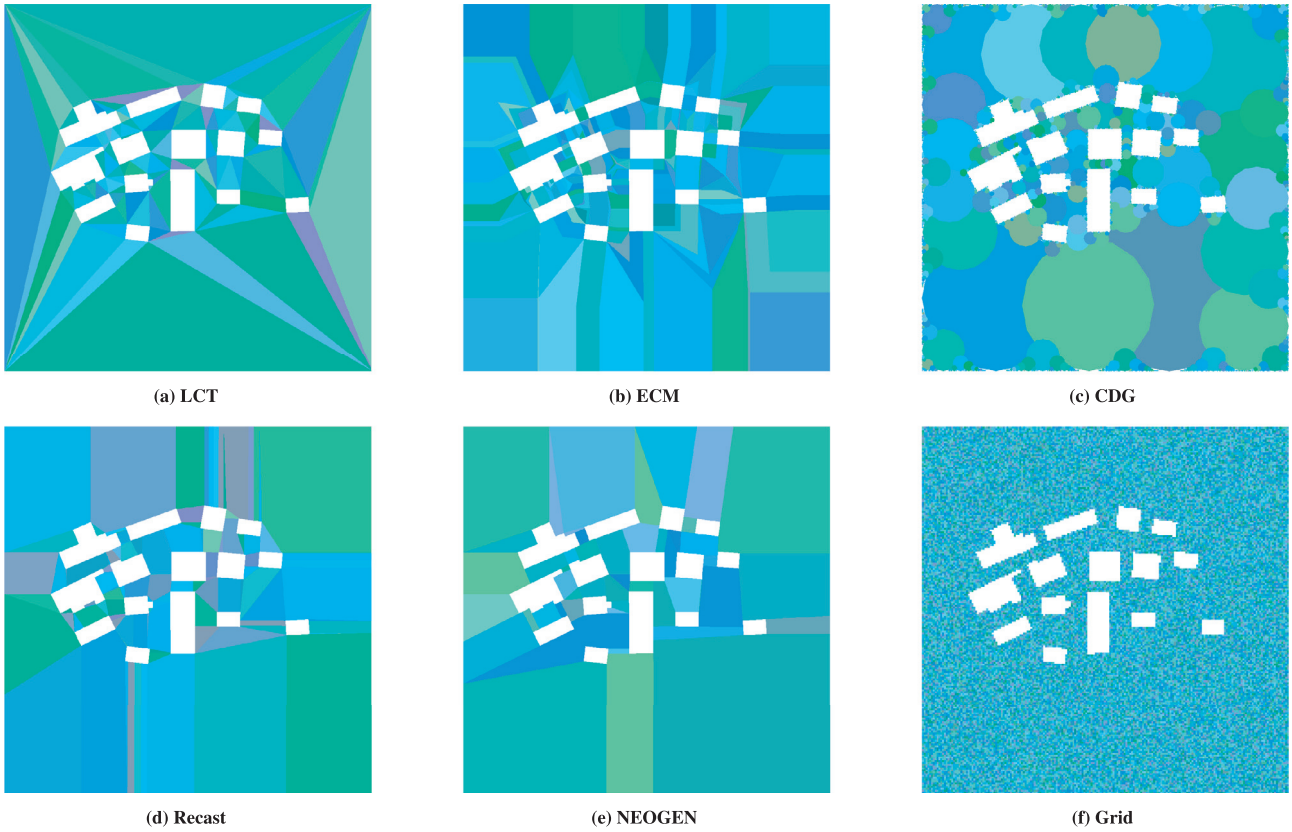


Fig. 12. Navigation meshes computed for the *Military* environment. Regions are shown in different colors.

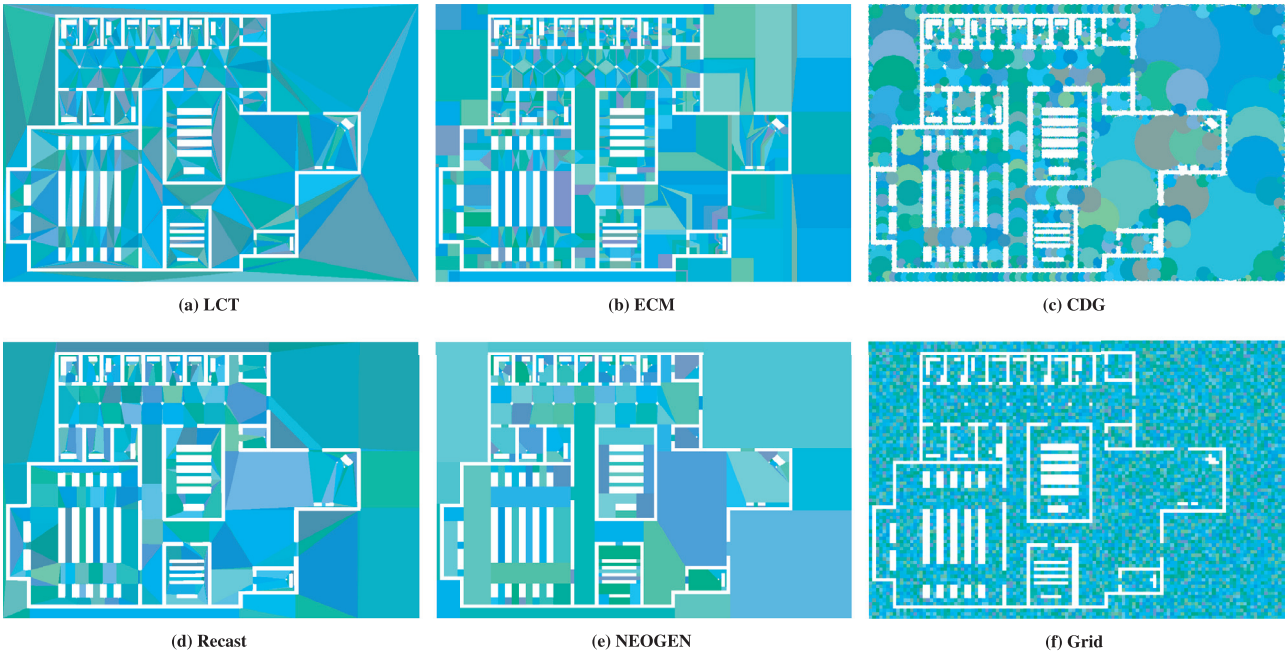


Fig. 13. Navigation meshes computed for the *University* environment. Regions are shown in different colors.

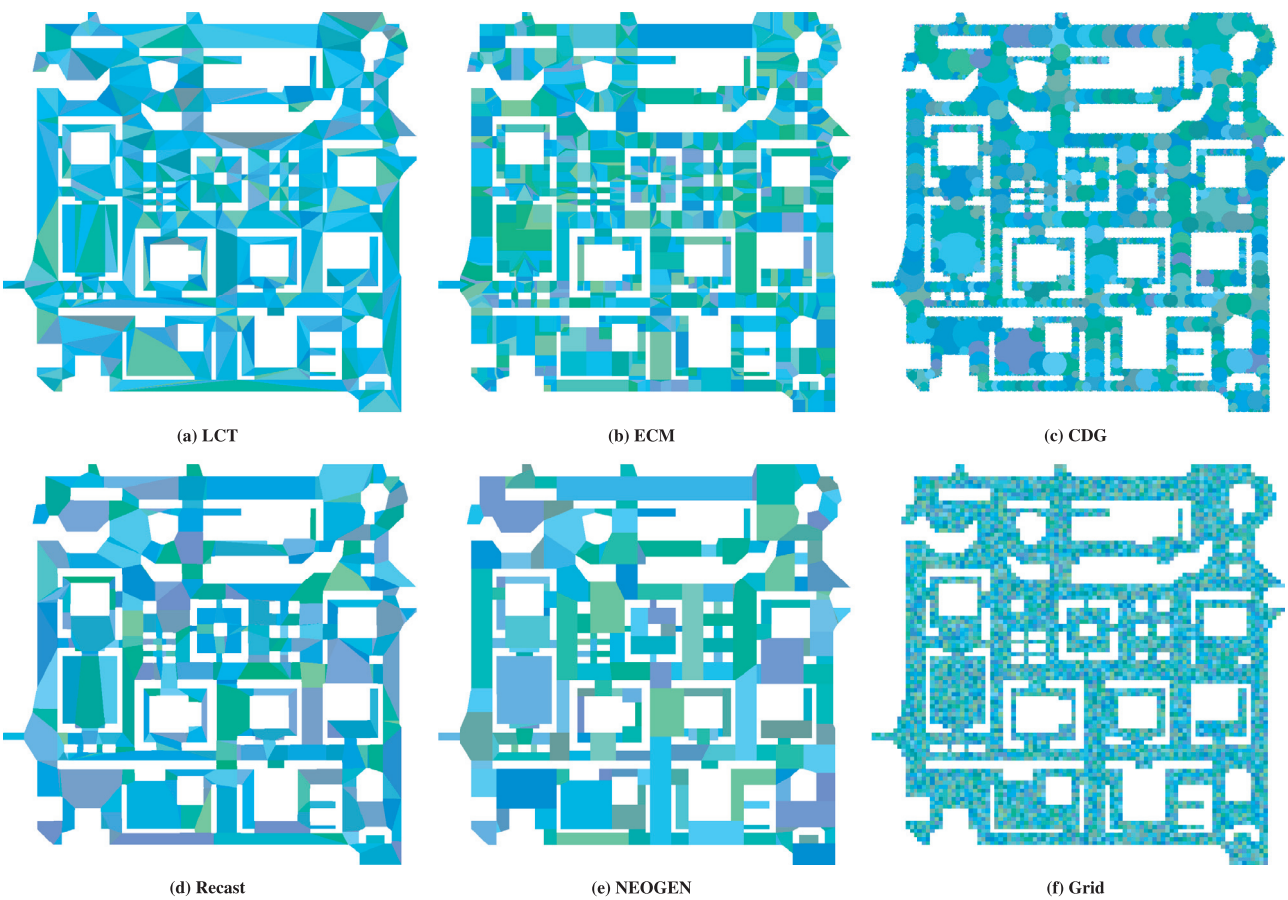


Fig. 14. Navigation meshes computed for the *Zelda* environment. Regions are shown in different colors.

7.3. Environments

We have computed navigation meshes for the 2D and 3D input environments shown in Fig. 9 and 10. Most of them have been used in previous publications or were included in one of the navigation mesh implementations. Furthermore, we have obtained various levels of the FPS game *Counter-Strike 1.6* [55] from an online repository [56]. Finally, to test for scalability, we have added randomly generated 2D mazes of various sizes, inspired by Sturtevant [34].

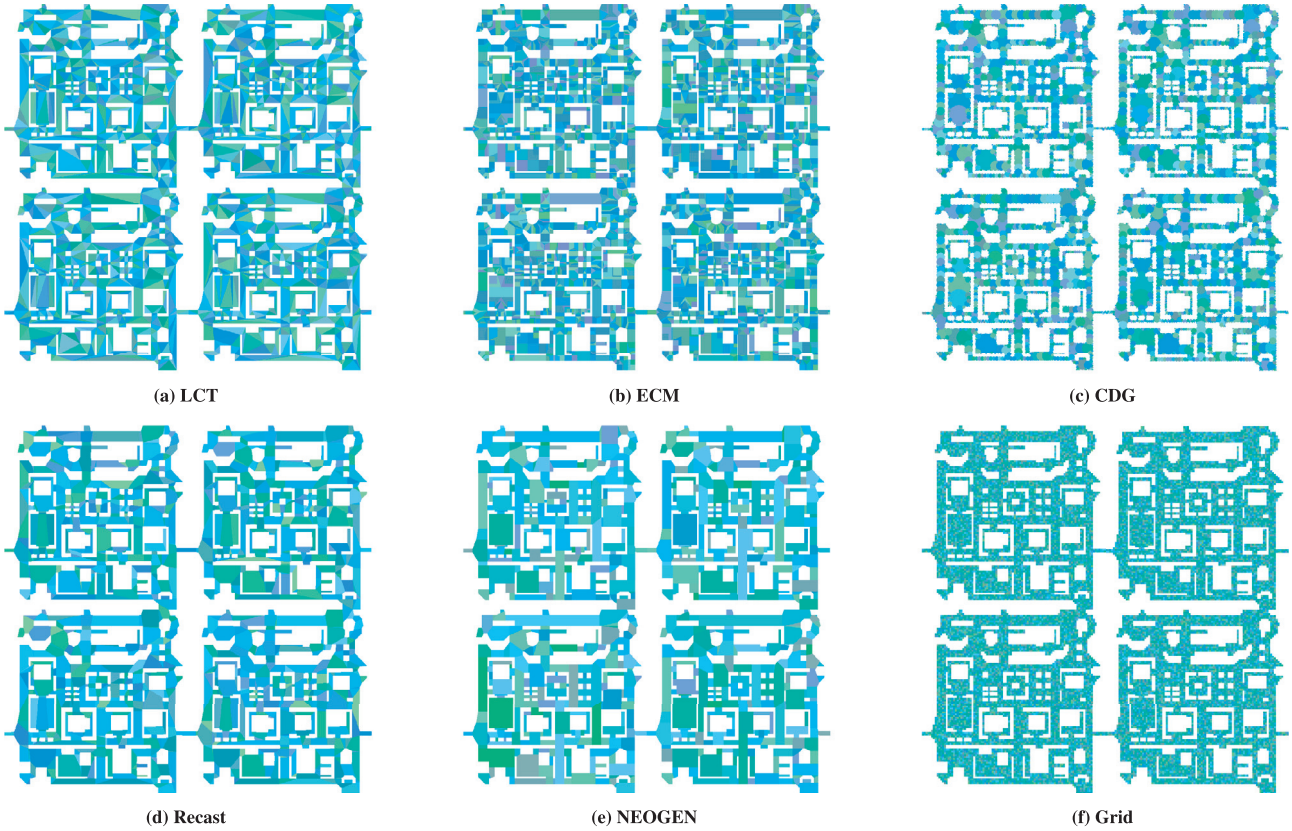


Fig. 15. Navigation meshes computed for the *Zelda2x2* environment. Regions are shown in different colors.

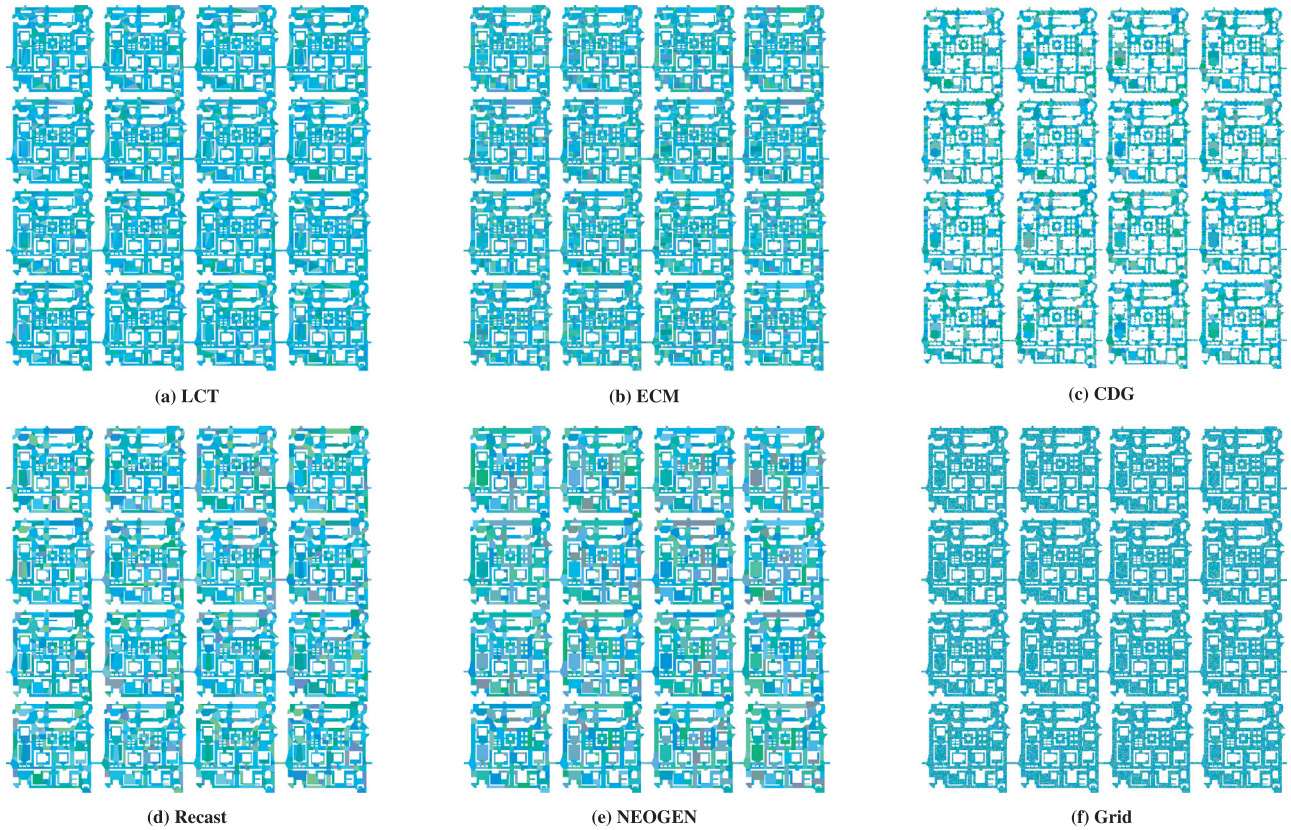
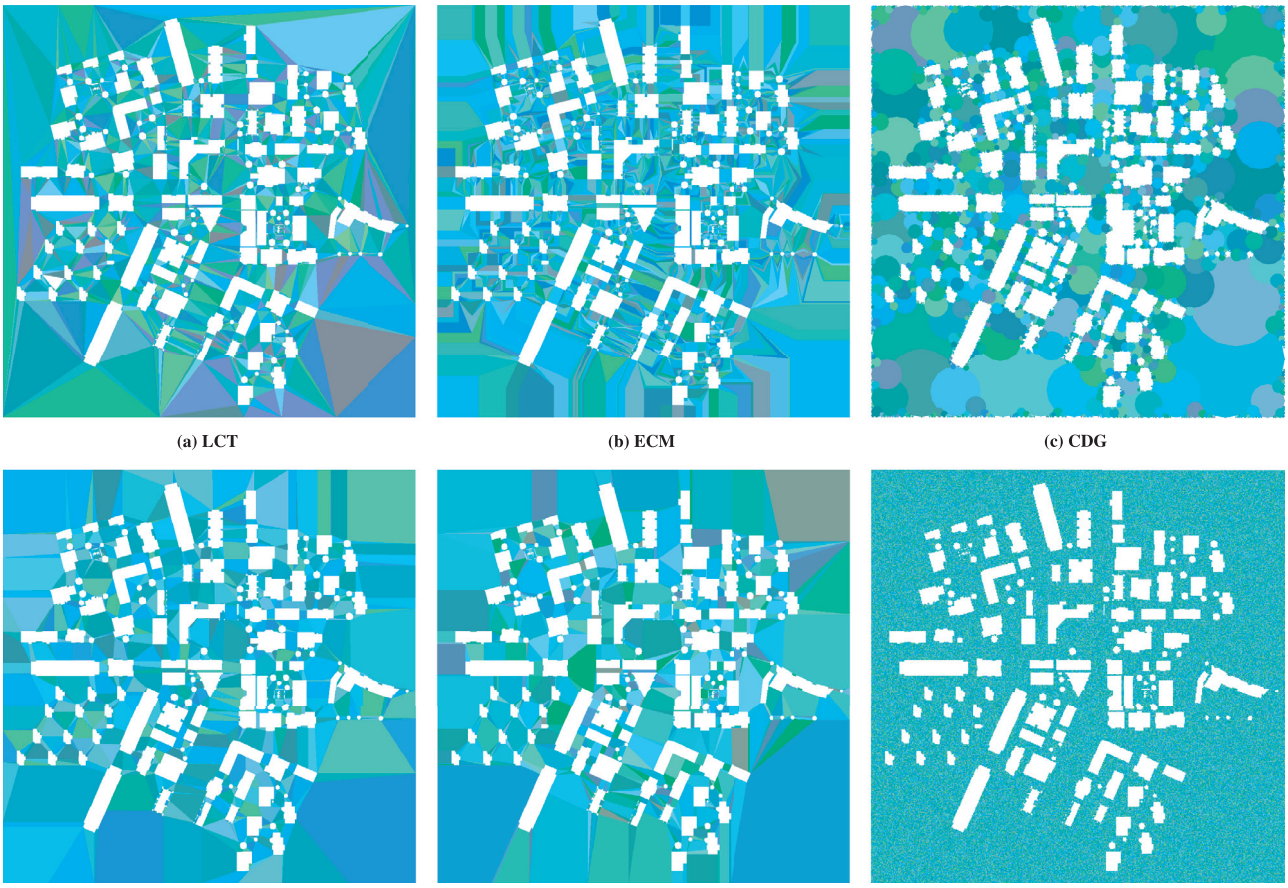


Fig. 16. Navigation meshes computed for the *Zelda4x4* environment. Regions are shown in different colors.



(a) LCT (b) ECM (c) CDG

(d) Recast (e) NEOGEN (f) Grid

Fig. 17. Navigation meshes computed for the *City* environment. Regions are shown in different colors.

(a) LCT (b) ECM (c) CDG (d) Recast (e) NEOGEN (f) Grid

Fig. 18. Navigation meshes computed for the *Maze8* environment. Regions are shown in different colors.

Using CGAL-based filtering software [17] and manual editing, we have converted each environment to a clean representation of $\mathcal{E}_{free}$, subdivided into layers when necessary.

7.4. Discussion of results

For our 2D environments, the navigation meshes produced by all algorithms are shown in Fig. 11–22. For the MLEs, the results are difficult to visualize in a single image; as a representative example, Fig. 23 shows the results for *Library*.

Tables 3 –10 contain the quantitative results. We have created separate tables for each category of metrics. We will now discuss the most important observations in all categories.

7.4.1. ***Coverage*** *(Tables 3 and 4).*

Despite our use of small voxels, the voxel-based methods sometimes missed large areas or covered large incorrect parts, up to hundreds of square meters in large enviroments. However, the *normalized* coverage was still high (typically over 90%). Interestingly, Recast always gave slightly 'distorted' output coordinates, even if the input was perfectly grid-aligned. This is easiest to see in the *Maze8* environment (Fig. 18d). (Decreasing the voxel size reduces this distortion, but it never fully disappears.) However, the relative coverage of Recast in the mazes remained decent at over 80%.

NEOGEN generally yielded high coverage due to its additional high-precision processing step per layer. This indicates that voxel-based methods can work well, given enough precision in the phase that converts voxels to polygons. It would be interesting to let methods automatically choose an appropriate resolution based on a user-specified balance between coverage and performance. A theoretically stronger alternative would be to reconstruct $\mathcal{E}_{free}$ without relying on a grid resolution.

For completeness, we have also calculated coverage for the *exact* methods. As expected, these methods usually scored nearly

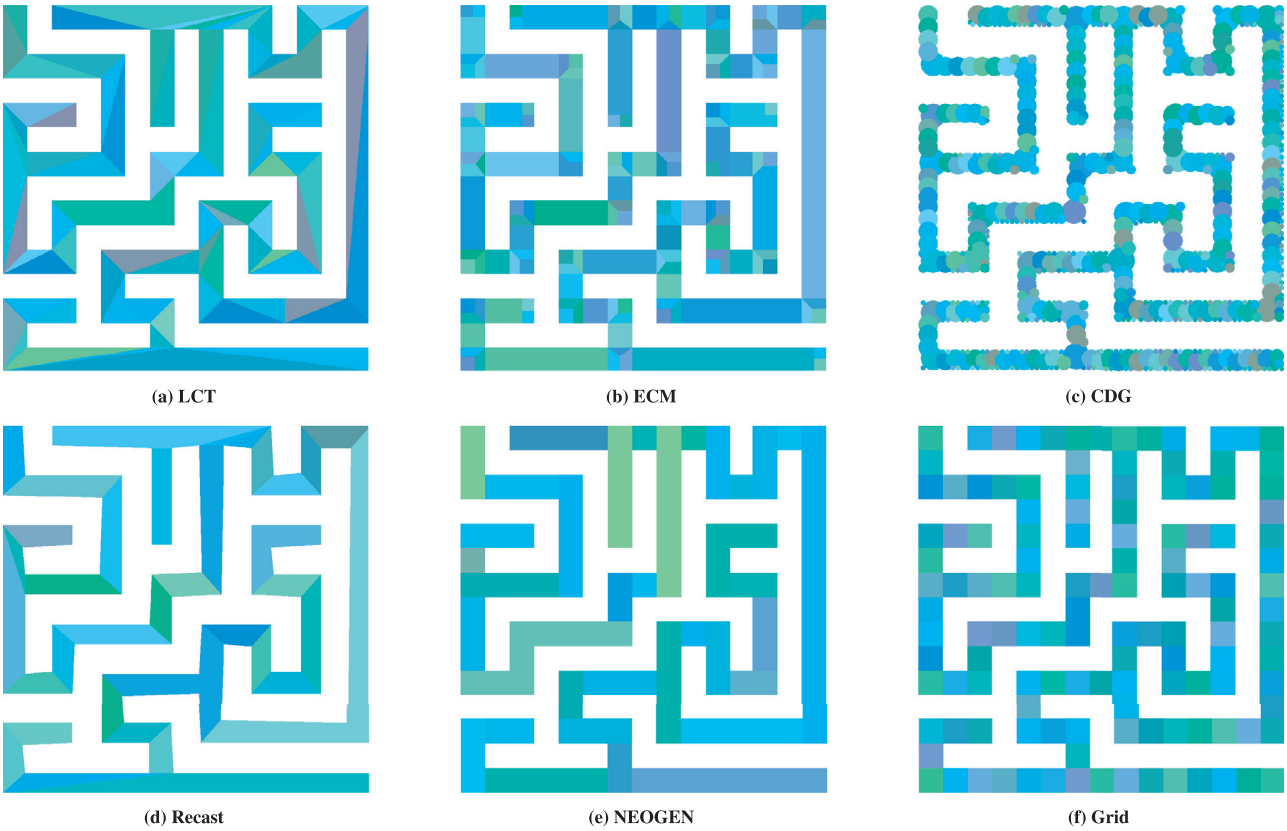


Fig. 19. Navigation meshes computed for the *Maze16* environment. Regions are shown in different colors.

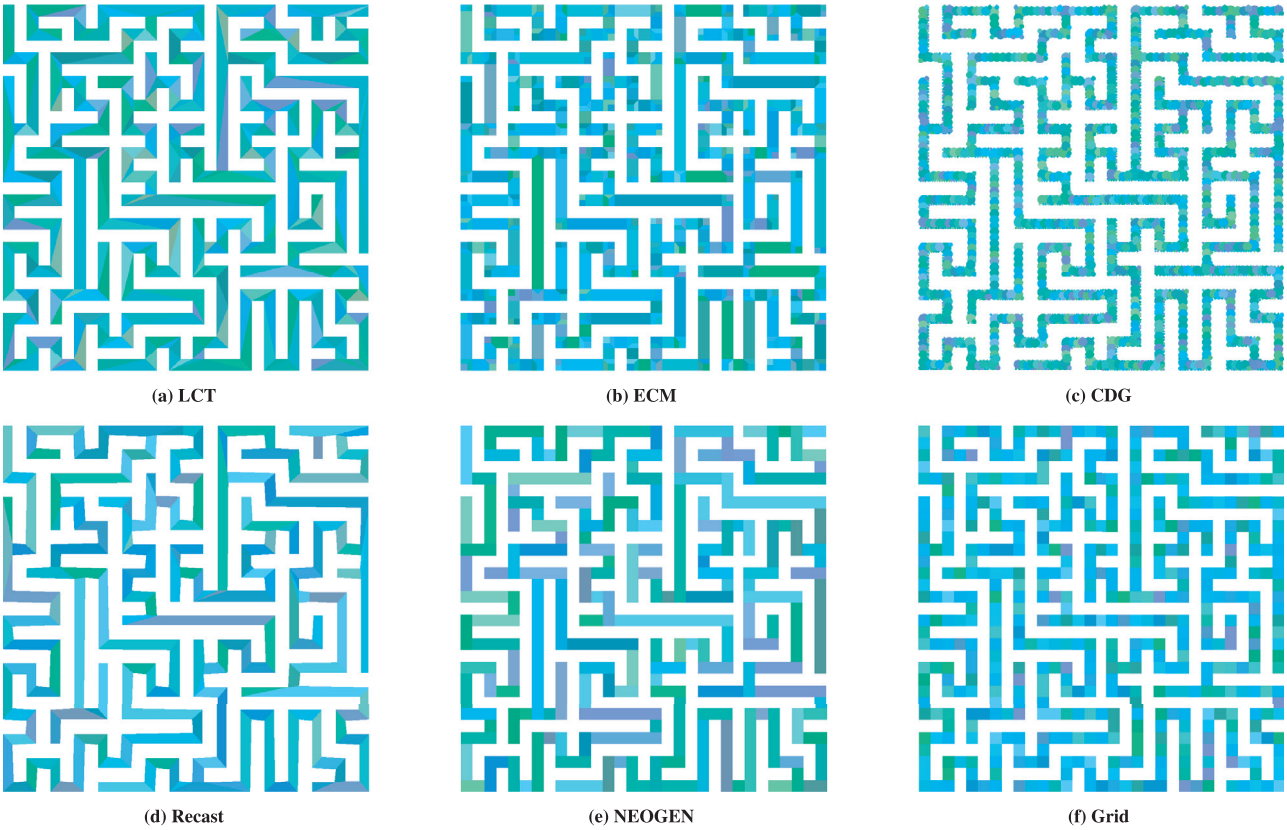


Fig. 20. Navigation meshes computed for the *Maze32* environment. Regions are shown in different colors.

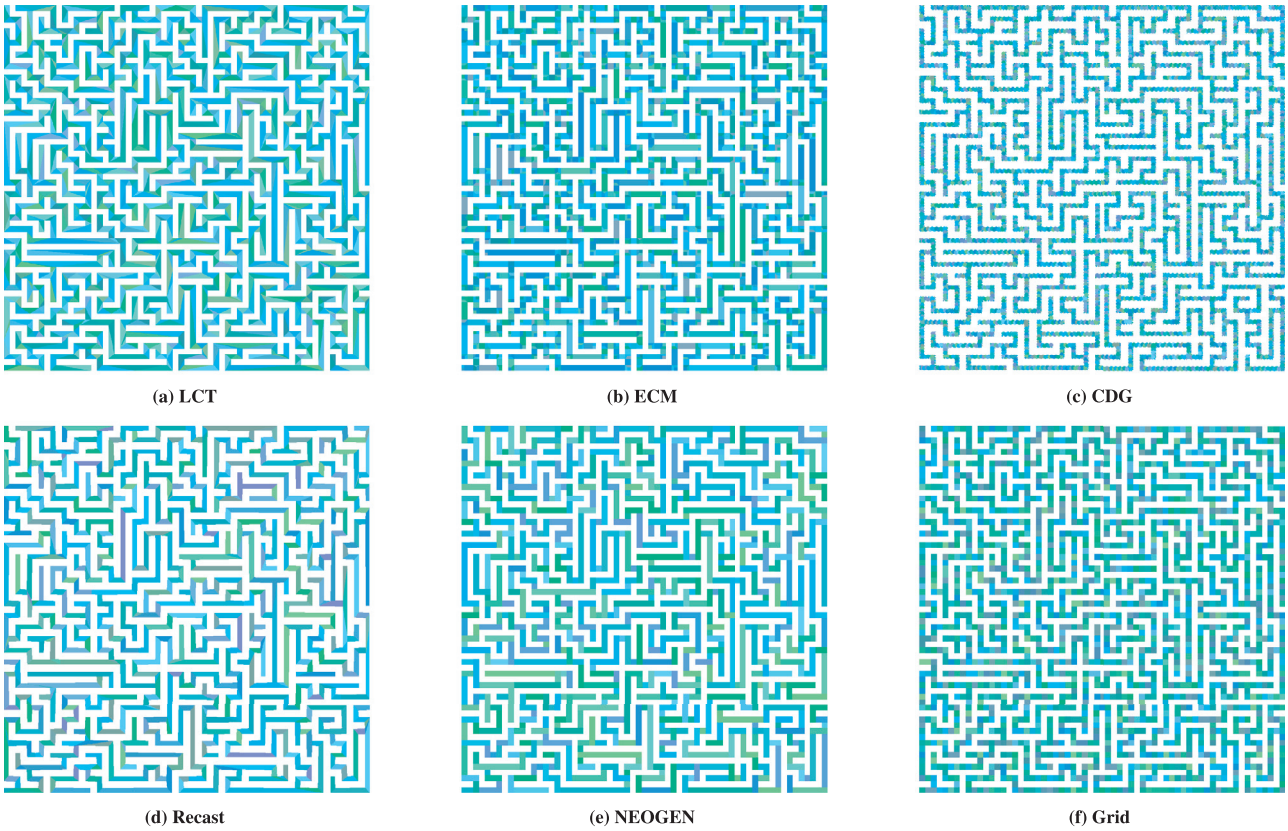


Fig. 21. Navigation meshes computed for the *Maze64* environment. Regions are shown in different colors.

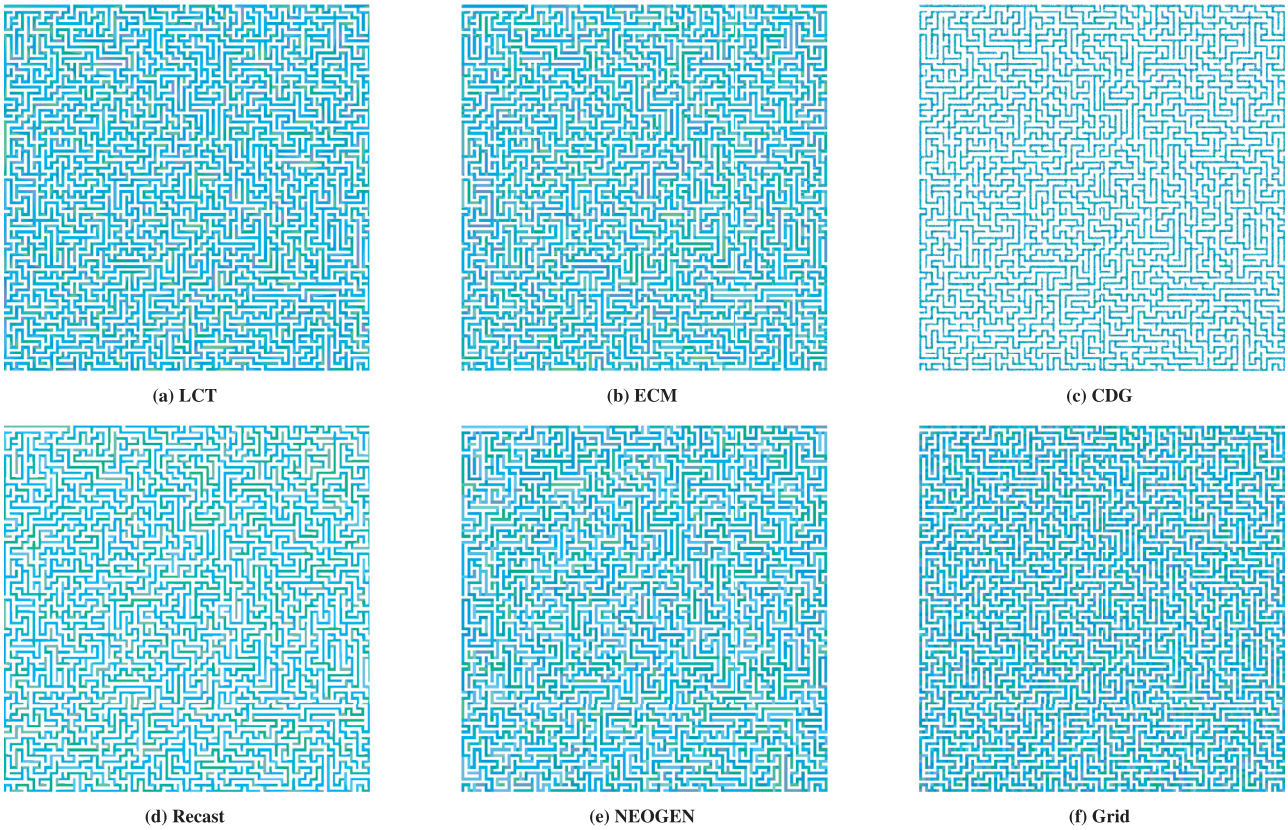


Fig. 22. Navigation meshes computed for the *Maze128* environment. Regions are shown in different colors.

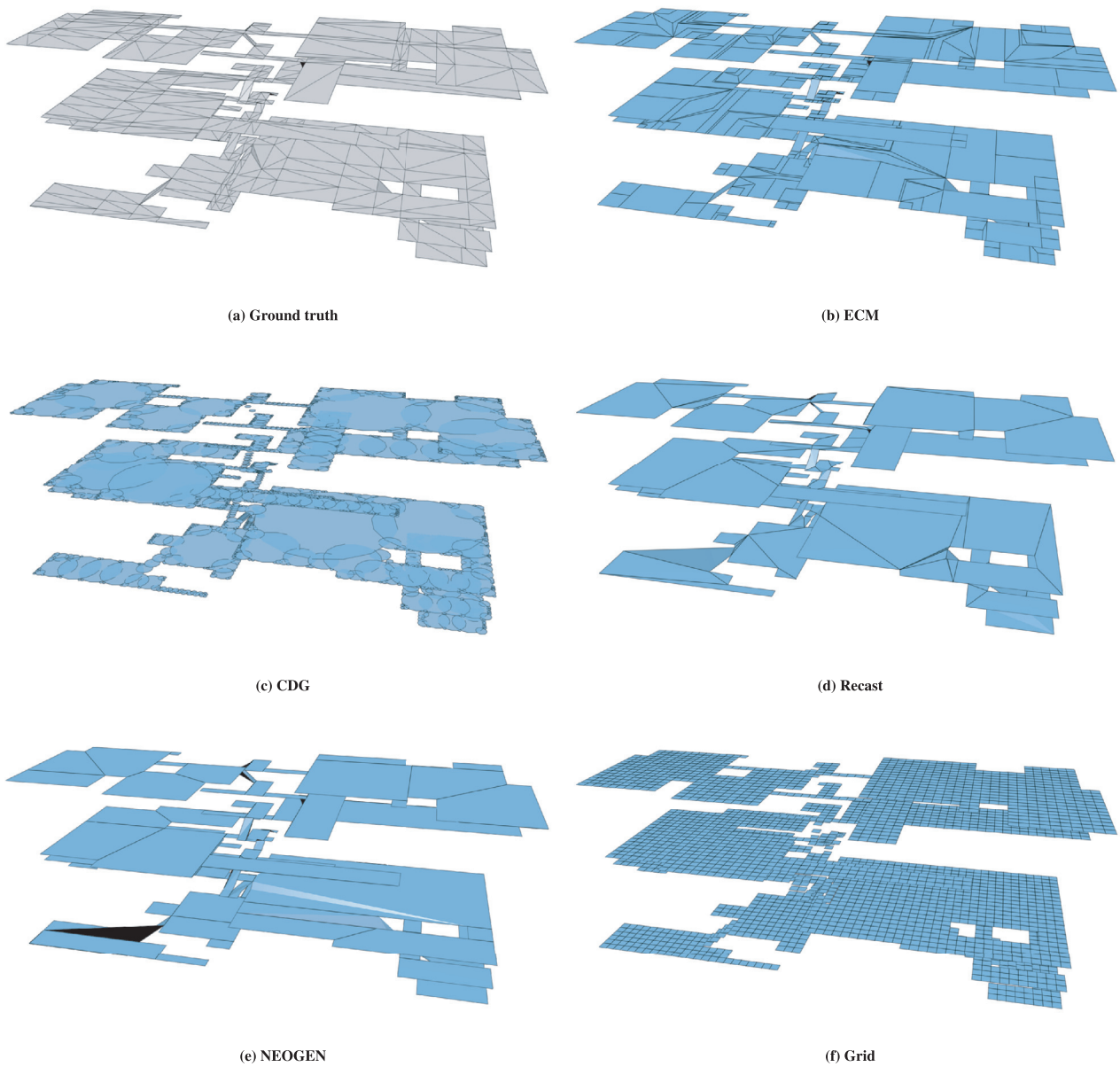


Fig. 23. Renderings of the navigation meshes computed for the *Library* environment. Regions are shown in blue and outlined in black. The corresponding graphs have been omitted for clarity. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

perfectly; any tiny deviations from a perfect score are caused by floating-point precision in our benchmark software. In the *BigCity* environment, the ECM could not produce any height coordinates, most likely due to an implementation error. This caused all navigation-mesh regions to be mapped onto the ground plane, leading to poor coverage results. However, upon inspection, the navigation mesh appears to be geometrically correct except for these height coordinates.

As explained in Section 7.1, we use a vertical threshold distance $T = 1$m to compute coverage. This can give incorrect results in MLEs with large height differences that are not represented by navigation-mesh regions. An example of this is the *Neogen1* environment (Fig. 10g) for which some navigation meshes do not capture the 'hill' shape in the bottom-left corner. This hill is then considered as 'not covered', even though there is clearly a corresponding (flat) region in the navigation mesh. Aside from exceptions like this one, our coverage calculations are sufficiently accurate in almost all environments.

7.4.2. ***Connectivity*** *(Tables 5 and 6).*

Recast and NEOGEN captured connectivity quite well for most environments, except in some of the more complex MLEs. For the CDG, the graph usually contained too many connected components, and gaps in the covered space led to a large number of boundaries.

Again, we have also included the results for the LCT and the ECM, for completeness. These methods usually yielded perfect connectivity values. There were a few exceptions for the LCT, most likely caused by our own process of converting $\mathcal{E}_{free}$ to a boundary representation.

It is motivating to see that any clearly bad connectivity values corresponded to navigation meshes that were also *visually* incorrect. For example, Recast accidentally connected layers vertically in the *Tower* environment (see Fig. 8), which led to a high value for the 'number of boundaries' metric. This suggests that our connectivity metrics are suitable for quickly detecting large errors in a navigation mesh. However, to find the exact nature of the error, vi-

Table 3
Results for the coverage metrics in the 2D environments. The descriptions of all metrics can be found in Section 6. Numbers in boldface indicate a perfect value.

Environment		Navigation mesh	Coverage					
	Total area (m^2)		Cov	Cov'	A_{inc}	A'_{inc}	Ov	Ov'
Simple	6,859.29	LCT	6,859.25	1.00	0.03	0.00	0.01	0.00
		ECM	6,859.26	1.00	0.03	0.00	**0.00**	0.00
		CDG	6,623.67	0.97	4.42	0.00	3,096.95	0.32
		Recast	6,776.79	0.99	1.46	0.00	**0.00**	0.00
		NEOGEN	6,859.26	1.00	0.03	0.00	**0.00**	0.00
		Grid	6,713.73	0.98	116.27	0.02	**0.00**	0.00
Military	36,876.82	LCT	36,876.81	1.00	**0.00**	0.00	0.01	0.00
		ECM	36,876.80	1.00	0.04	0.00	**0.00**	0.00
		CDG	36,207.40	0.98	7.99	0.00	15,604.37	0.30
		Recast	36,752.50	1.00	1.66	0.00	**0.00**	0.00
		NEOGEN	36,876.83	1.00	**0.00**	0.00	**0.00**	0.00
		Grid	36,772.79	1.00	117.21	0.00	0.01	0.00
University	8,370.68	LCT	8,370.63	1.00	0.04	0.00	2.41	0.00
		ECM	**8,370.68**	1.00	**0.00**	0.00	**0.00**	0.00
		CDG	7,823.18	0.93	11.45	0.00	3,385.77	0.30
		Recast	8,206.46	0.98	9.60	0.00	0.01	0.00
		NEOGEN	**8,370.68**	1.00	**0.00**	0.00	**0.00**	0.00
		Grid	7,982.70	0.95	578.56	0.07	**0.00**	0.00
Zelda	5,642.24	LCT	**5,642.24**	1.00	**0.00**	0.00	**0.00**	0.00
		ECM	5,642.23	1.00	0.01	0.00	**0.00**	0.00
		CDG	5,269.79	0.93	9.04	0.00	2,343.15	0.31
		Recast	5,453.28	0.97	0.67	0.00	**0.00**	0.00
		NEOGEN	**5,642.24**	1.00	**0.00**	0.00	**0.00**	0.00
		Grid	5,480.73	0.97	55.27	0.01	**0.00**	0.00
Zelda2x2	22,632.42	LCT	**22,632.42**	1.00	**0.00**	0.00	**0.00**	0.00
		ECM	22,632.38	1.00	0.08	0.00	**0.00**	0.00
		CDG	19,369.98	0.86	85.94	0.00	6,784.67	0.26
		Recast	21,878.87	0.97	2.31	0.00	**0.00**	0.00
		NEOGEN	**22,632.42**	1.00	**0.00**	0.00	**0.00**	0.00
		Grid	21,978.13	0.97	221.87	0.01	**0.00**	0.00
Zelda4x4	90,529.70	LCT	90,529.69	1.00	**0.00**	0.00	**0.00**	0.00
		ECM	90,529.55	1.00	0.18	0.00	**0.00**	0.00
		CDG	65,836.02	0.73	844.11	0.01	16,331.52	0.20
		Recast	87,794.71	0.97	32.48	0.00	**0.00**	0.00
		NEOGEN	90,529.69	1.00	0.01	0.00	**0.00**	0.00
		Grid	87,912.54	0.97	887.45	0.01	0.01	0.00
City	207,518.40	LCT	207,284.10	1.00	0.11	0.00	0.10	0.00
		ECM	207,518.10	1.00	0.28	0.00	0.03	0.00
		CDG	197,066.40	0.95	302.06	0.00	82,839.06	0.30
		Recast	206,250.00	0.99	80.04	0.00	**0.00**	0.00
		NEOGEN	207,518.30	1.00	0.13	0.00	**0.00**	0.00
		Grid	206,227.00	0.99	1,255.12	0.01	**0.00**	0.00
Maze8	31.00	LCT	**31.00**	1.00	**0.00**	0.00	**0.00**	0.00
		ECM	**31.00**	1.00	**0.00**	0.00	**0.00**	0.00
		CDG	24.06	0.78	**0.00**	0.00	8.54	0.26
		Recast	25.58	0.83	**0.00**	0.00	**0.00**	0.00
		NEOGEN	**31.00**	1.00	**0.00**	0.00	**0.00**	0.00
		Grid	**31.00**	1.00	**0.00**	0.00	**0.00**	0.00
Maze16	127.00	LCT	**127.00**	1.00	**0.00**	0.00	**0.00**	0.00
		ECM	**127.00**	1.00	**0.00**	0.00	**0.00**	0.00
		CDG	101.74	0.80	0.03	0.00	46.18	0.31
		Recast	106.14	0.84	**0.00**	0.00	**0.00**	0.00
		NEOGEN	**127.00**	1.00	**0.00**	0.00	**0.00**	0.00
		Grid	**127.00**	1.00	**0.00**	0.00	**0.00**	0.00
Maze32	511.00	LCT	**511.00**	1.00	**0.00**	0.00	**0.00**	0.00
		ECM	**511.00**	1.00	**0.00**	0.00	**0.00**	0.00
		CDG	408.38	0.80	0.15	0.00	171.84	0.30
		Recast	427.98	0.84	0.02	0.00	**0.00**	0.00
		NEOGEN	**511.00**	1.00	**0.00**	0.00	**0.00**	0.00
		Grid	**511.00**	1.00	**0.00**	0.00	**0.00**	0.00
Maze64	2,047.00	LCT	**2,047.00**	1.00	**0.00**	0.00	**0.00**	0.00
		ECM	**2,047.00**	1.00	**0.00**	0.00	**0.00**	0.00
		CDG	1,582.16	0.77	7.14	0.00	601.59	0.28
		Recast	1,716.27	0.84	**0.00**	0.00	**0.00**	0.00
		NEOGEN	**2,047.00**	1.00	**0.00**	0.00	**0.00**	0.00
		Grid	**2,047.00**	1.00	**0.00**	0.00	**0.00**	0.00

(*continued on next page*)

Table 3 (*continued*)

Environment	Total area (m^2)	Navigation mesh	Coverage					
			Cov	Cov'	A_{inc}	A'_{inc}	Ov	Ov'
Maze128	8,191.00	LCT	**8,191.00**	1.00	**0.00**	0.00	**0.00**	0.00
		ECM	**8,191.00**	1.00	**0.00**	0.00	**0.00**	0.00
		CDG	5,067.01	0.62	109.55	0.02	896.18	0.15
		Recast	6,861.23	0.84	**0.00**	0.00	**0.00**	0.00
		NEOGEN	**8,191.00**	1.00	**0.00**	0.00	**0.00**	0.00
		Grid	**8,191.00**	1.00	**0.00**	0.00	**0.00**	0.00

Table 4
Results for the coverage metrics in the multi-layered environments. An empty row indicates that the navigation mesh could not be computed for the corresponding algorithm and environment. Numbers in boldface indicate a perfect value.

Environment	Total area (m^2)	Navigation mesh	Coverage					
			Cov	Cov'	A_{inc}	A'_{inc}	Ov	Ov'
as_oilrig	75,746.52	ECM	74,976.05	0.99	766.87	0.01	3.55	0.00
		CDG	-	-	-	-	-	-
		Recast	75,302.81	0.99	18.62	0.00	5.41	0.00
		NEOGEN	-	-	-	-	-	-
		Grid	74,772.91	0.99	1,389.73	0.02	0.37	0.00
cs_assault	21,779.29	ECM	19,475.67	0.89	2,303.61	0.12	**0.00**	0.00
		CDG	19,190.24	0.88	2,599.88	0.09	8,530.28	0.31
		Recast	21,531.92	0.99	32.14	0.00	0.03	0.00
		NEOGEN	17,998.92	0.83	2,934.28	0.16	0.01	0.00
		Grid	21,110.20	0.97	726.38	0.03	**0.00**	0.00
cs_siege	26,207.49	ECM	26,173.85	1.00	33.61	0.00	0.05	0.00
		CDG	23,521.03	0.90	507.83	0.02	9,959.50	0.30
		Recast	25,899.19	0.99	13.82	0.00	0.55	0.00
		NEOGEN	24,671.52	0.94	1,404.44	0.06	3.81	0.00
		Grid	25,649.85	0.98	628.54	0.02	**0.00**	0.00
de_dust	25,106.82	ECM	24,755.99	0.99	350.81	0.01	**0.00**	0.00
		CDG	22,954.53	0.91	1,217.82	0.04	8,787.81	0.28
		Recast	24,824.24	0.99	6.10	0.00	0.01	0.00
		NEOGEN	24,920.51	0.99	185.82	0.01	0.48	0.00
		Grid	24,571.73	0.98	600.34	0.02	**0.00**	0.00
de_dust2	20,243.71	ECM	20,172.35	1.00	71.36	0.00	**0.00**	0.00
		CDG	18,589.43	0.92	1,547.66	0.06	7,214.72	0.28
		Recast	20,036.42	0.99	7.53	0.00	**0.00**	0.00
		NEOGEN	19,869.34	0.98	357.22	0.02	17.14	0.00
		Grid	19,804.10	0.98	650.92	0.03	**0.00**	0.00
Jardin	2,802.35	ECM	2,651.16	0.95	143.28	0.05	7.94	0.00
		CDG	2,447.30	0.87	432.51	0.13	1,011.72	0.29
		Recast	2,750.82	0.98	9.47	0.00	0.68	0.00
		NEOGEN	2,246.60	0.80	356.63	0.16	7.54	0.00
		Grid	2,744.80	0.98	73.70	0.03	**0.00**	0.00
Neogen1	4,748.47	ECM	4,671.50	0.98	73.46	0.02	3.48	0.00
		CDG	4,389.95	0.92	565.87	0.09	1,747.39	0.28
		Recast	4,708.47	0.99	2.66	0.00	0.33	0.00
		NEOGEN	4,429.13	0.93	223.37	0.05	89.10	0.02
		Grid	4,691.59	0.99	137.75	0.03	4.66	0.00
Neogen2	9,600.56	ECM	9,600.43	1.00	0.17	0.00	0.02	0.00
		CDG	9,277.86	0.97	301.94	0.02	4,220.22	0.31
		Recast	9,518.50	0.99	8.46	0.00	2.68	0.00
		NEOGEN	9,371.74	0.98	20.84	0.00	**0.00**	0.00
		Grid	9,414.95	0.98	253.79	0.03	1.27	0.00
Neogen3	9,642.51	ECM	9,642.28	1.00	0.13	0.00	**0.00**	0.00
		CDG	9,444.76	0.98	39.43	0.00	3,920.22	0.29
		Recast	9,554.90	0.99	6.27	0.00	**0.00**	0.00
		NEOGEN	9,527.22	0.99	15.10	0.00	**0.00**	0.00
		Grid	9,494.22	0.98	288.90	0.03	**0.00**	0.00
Dungeon	2,114.80	ECM	2,114.71	1.00	0.07	0.00	0.01	0.00
		CDG	1,958.18	0.93	201.40	0.08	599.81	0.23
		Recast	2,087.57	0.99	7.25	0.00	**0.00**	0.00

(*continued on next page*)

Table 4 (*continued*)

Environment		Navigation mesh	Coverage					
	Total area (m^2)		Cov	Cov'	A_{inc}	A'_{inc}	Ov	Ov'
		NEOGEN	-	-	-	-	-	-
		Grid	2,016.20	0.95	115.98	0.06	**0.00**	0.00
NavTest	5,202.13	ECM	5,199.64	1.00	2.33	0.00	**0.00**	0.00
		CDG	4,632.57	0.89	681.27	0.11	1,671.97	0.27
		Recast	5,133.04	0.99	3.22	0.00	0.01	0.00
		NEOGEN	4,532.81	0.87	450.52	0.10	**0.00**	0.00
		Grid	4,971.44	0.96	310.22	0.06	**0.00**	0.00
ParkingLot	1,921.50	ECM	1,921.39	1.00	0.10	0.00	0.02	0.00
		CDG	1,842.43	0.96	0.13	0.00	750.19	0.29
		Recast	1,890.67	0.98	2.14	0.00	0.05	0.00
		NEOGEN	-	-	-	-	-	-
		Grid	1,884.49	0.98	203.28	0.11	6.25	0.00
Library	3,154.06	ECM	3,153.85	1.00	0.10	0.00	0.02	0.00
		CDG	3,023.70	0.96	9.22	0.00	1,168.79	0.28
		Recast	3,094.35	0.98	4.06	0.00	0.07	0.00
		NEOGEN	3,132.67	0.99	21.39	0.01	**0.00**	0.00
		Grid	2,986.48	0.95	201.53	0.07	**0.00**	0.00
Tower	12,093.88	ECM	12,092.38	1.00	1.43	0.00	**0.00**	0.00
		CDG	10,997.80	0.91	24.59	0.00	3,606.29	0.25
		Recast	11,840.82	0.98	288.76	0.02	0.69	0.00
		NEOGEN	-	-	-	-	-	-
		Grid	11,367.29	0.94	1,074.77	0.09	**0.00**	0.00
BigCity	280,876.10	ECM	191,738.30	0.68	40,652.61	0.17	48,505.70	0.20
		CDG	-	-	-	-	-	-
		Recast	277,278.10	0.99	2,787.79	0.01	**0.00**	0.00
		NEOGEN	-	-	-	-	-	-
		Grid	-	-	-	-	-	-

Table 5
Results for the connectivity metrics in the 2D environments. '#CCs' is an abbreviation of 'number of connected components'. The descriptions of all metrics can be found in Section 6. Numbers in boldface indicate a perfect value.

Environment			Navigation mesh	Connectivity	
	#CCs	#Boundaries		#CCs	#Boundaries
Simple	1	3	LCT	**1**	**3**
			ECM	**1**	**3**
			CDG	2	1,329
			Recast	**1**	**3**
			NEOGEN	4	6
			Grid	**1**	**3**
Military	1	16	LCT	**1**	**16**
			ECM	**1**	**16**
			CDG	3	906
			Recast	**1**	**16**
			NEOGEN	**1**	**16**
			Grid	**1**	**16**
University	1	82	LCT	2	86
			ECM	**1**	**82**
			CDG	274	1,391
			Recast	**1**	**82**
			NEOGEN	4	83
			Grid	**1**	53
Zelda	1	57	LCT	**1**	**57**
			ECM	**1**	**57**
			CDG	3	1,313
			Recast	**1**	**57**
			NEOGEN	**1**	**57**
			Grid	**1**	**57**
Zelda2x2	1	226	LCT	**1**	**226**
			ECM	**1**	**226**

(*continued on next page*)

Table 5 (*continued*)

Environment			Navigation mesh	Connectivity	
	#CCs	#Boundaries		#CCs	#Boundaries
			CDG	9	3,088
			Recast	**1**	**226**
			NEOGEN	**1**	**226**
			Grid	**1**	**226**
Zelda4x4	1	906	LCT	**1**	**906**
			ECM	**1**	**906**
			CDG	244	4,928
			Recast	**1**	**906**
			NEOGEN	**1**	**906**
			Grid	**1**	**906**
City	1	181	LCT	2	178
			ECM	**1**	**181**
			CDG	203	2,209
			Recast	**1**	183
			NEOGEN	12	177
			Grid	2	184
Maze8	1	1	LCT	**1**	**1**
			ECM	**1**	**1**
			CDG	**1**	41
			Recast	**1**	**1**
			NEOGEN	**1**	**1**
			Grid	**1**	**1**
Maze16	1	1	LCT	**1**	**1**
			ECM	**1**	**1**
			CDG	**1**	272
			Recast	**1**	**1**
			NEOGEN	**1**	**1**
			Grid	**1**	**1**
Maze32	1	1	LCT	**1**	**1**
			ECM	**1**	**1**
			CDG	**1**	1,021
			Recast	**1**	**1**
			NEOGEN	**1**	**1**
			Grid	**1**	**1**
Maze64	1	1	LCT	**1**	**1**
			ECM	**1**	**1**
			CDG	289	5,434
			Recast	**1**	**1**
			NEOGEN	**1**	**1**
			Grid	**1**	**1**
Maze128	1	1	LCT	**1**	**1**
			ECM	**1**	**1**
			CDG	1188	22,151
			Recast	**1**	**1**
			NEOGEN	**1**	**1**
			Grid	**1**	**1**

sual inspection is still needed. (In the example of Fig. 8, decreasing the voxel size did not solve the problem. The issue is most likely caused by our use of a character radius of zero, a setting for which Recast does not seem to be designed.)

7.4.3. ***Complexity*** *(Tables 7 and 8).*

NEOGEN and Recast typically yielded small graphs and a low total region complexity. These methods produce compact descriptions of the environment, thanks to their extra processing steps for grouping voxels into polygons. This suggests that voxel-based methods are (currently) ideal whenever a low memory footprint and fast path planning are more important than coverage.

The ECM often produced smaller graphs than the LCT, in exchange for a higher total region complexity. Both methods usually yielded more complex meshes than NEOGEN or Recast. This is because exact methods capture all details of the environment, whereas voxel-based methods inherently simplify $\mathcal{E}_{free}$ to some extent. In future research, exact methods could *also* benefit from such a simplification, to prevent them from 'getting lost' in tiny details.

It should be noted that Recast contains several parameters (e.g. the maximum number of vertices per region) that allows users to adjust the balance between graph complexity and region complexity. The ideal balance depends heavily on the application. With our current settings, though, Recast already scored well in both aspects.

As expected, our grid implementation usually gave the largest graph, except in some of the mazes. This confirms that grids are usually inefficient representations, although we acknowledge their ease of use and their attractiveness for grid-aligned applications.

7.4.4. ***Performance*** *(Tables 9 and 10).*

The LCT implementation was the fastest in all environments, although it did require pre-processing (i.e. conversion to a boundary representation) that we have not included in our measurements. A similar conversion *is* included in the ECM implementation, which makes the two a bit difficult to compare in terms of performance.

Table 6
Results for the connectivity metrics in the multi-layered environments. '#CCs' is an abbreviation of 'number of connected components'. An empty row indicates that the navigation mesh could not be computed for the corresponding algorithm and environment. Numbers in boldface indicate a perfect value.

Environment			Navigation mesh	Connectivity	
	#CCs	#Boundaries		#CCs	#Boundaries
as_oilrig	1	26	ECM	**1**	**26**
			CDG	-	-
			Recast	**1**	29
			NEOGEN	-	-
			Grid	**1**	21
cs_assault	10	24	ECM	**10**	**24**
			CDG	647	1,580
			Recast	7	43
			NEOGEN	**10**	**24**
			Grid	6	18
cs_siege	11	39	ECM	**11**	**39**
			CDG	501	1,214
			Recast	9	34
			NEOGEN	14	41
			Grid	13	31
de_dust	3	14	ECM	**3**	15
			CDG	452	1,146
			Recast	**3**	13
			NEOGEN	5	15
			Grid	4	17
de_dust2	1	6	ECM	**1**	**6**
			CDG	363	1,267
			Recast	2	5
			NEOGEN	9	12
			Grid	3	16
Jardin	12	33	ECM	13	35
			CDG	80	1,074
			Recast	9	28
			NEOGEN	1	22
			Grid	15	56
Neogen1	3	12	ECM	**3**	**12**
			CDG	91	1,253
			Recast	**3**	**12**
			NEOGEN	**3**	11
			Grid	**3**	14
Neogen2	11	33	ECM	**11**	**33**
			CDG	314	1,938
			Recast	10	31
			NEOGEN	13	32
			Grid	14	**33**
Neogen3	9	19	ECM	10	20
			CDG	60	2,345
			Recast	**9**	**19**
			NEOGEN	10	20
			Grid	22	42
Dungeon	10	12	ECM	**10**	**12**
			CDG	166	555
			Recast	5	9
			NEOGEN	-	-
			Grid	16	21
NavTest	23	43	ECM	**23**	**43**
			CDG	374	1,398
			Recast	22	39
			NEOGEN	9	29
			Grid	21	36
ParkingLot	1	9	ECM	**1**	**9**
			CDG	**1**	574
			Recast	**1**	**9**
			NEOGEN	-	-
			Grid	**1**	15

(*continued on next page*)

Table 6 (*continued*)

Environment			Navigation mesh	Connectivity	
	#CCs	#Boundaries		#CCs	#Boundaries
Library	1	4	ECM	**1**	**4**
			CDG	301	1,549
			Recast	**1**	**4**
			NEOGEN	6	8
			Grid	**1**	**4**
Tower	1	19	ECM	**1**	**19**
			CDG	6	9,770
			Recast	**1**	208
			NEOGEN	-	-
			Grid	**1**	195
BigCity	1	305	ECM	**1**	**305**
			CDG	-	-
			Recast	3	1,790
			NEOGEN	-	-
			Grid	-	-

Table 7
Results for the complexity metrics in the 2D environments. The descriptions of all metrics can be found in Section 6. Numbers in boldface indicate the lowest metric value (among all methods) for a particular environment.

Environment	Navigation mesh	Complexity					
					Region complexity		
		$\|V\|$	$\|E\|$	$\|\mathcal{R}\|$	Average	SD	Total
Simple	LCT	111	112	111	9.00	0.00	999
	ECM	94	95	191	14.81	2.29	2,829
	CDG	1746	3107	1746	**4.00**	0.00	6,984
	Recast	84	85	84	12.00	2.58	1,008
	NEOGEN	**44**	**42**	**44**	15.34	3.52	**675**
	Grid	6830	12,971	6830	12.00	0.00	81,960
Military	LCT	123	137	123	9.00	0.00	1,107
	ECM	58	72	214	14.83	2.16	3,174
	CDG	1168	2078	1168	**4.00**	0.00	4,672
	Recast	124	138	124	11.98	2.86	1,485
	NEOGEN	**52**	**66**	**52**	15.40	3.58	**801**
	Grid	36,890	72,851	36,890	12.00	0.00	442,680
University	LCT	738	818	738	9.00	0.00	6,642
	ECM	329	409	1134	15.04	2.29	17,055
	CDG	3308	4369	3308	**4.00**	0.00	13,232
	Recast	428	508	428	12.19	2.91	5,217
	NEOGEN	**261**	**334**	**261**	16.80	8.13	**4,386**
	Grid	8518	15,718	8518	12.00	0.00	102,216
Zelda	LCT	553	608	553	9.00	0.00	4,977
	ECM	289	344	895	14.93	2.30	13,359
	CDG	3381	4786	3381	**4.00**	0.00	13,524
	Recast	343	398	343	12.41	2.84	4,257
	NEOGEN	**204**	**259**	**204**	16.09	6.25	**3,282**
	Grid	5536	9895	5536	12.00	0.00	66,432
Zelda2x2	LCT	2250	2474	2250	9.00	0.00	20,250
	ECM	1148	1372	3602	14.92	2.30	53,754
	CDG	5636	8850	5636	**4.00**	0.00	22,544
	Recast	1345	1569	1345	12.48	2.85	16,782
	NEOGEN	**822**	**1,046**	**822**	16.13	6.31	**13,260**
	Grid	22,200	39,678	22,200	12.00	0.00	266,400
Zelda4x4	LCT	9004	9908	9004	9.00	0.00	81,036
	ECM	4580	5484	14,436	14.92	2.30	215,424
	CDG	11,996	16,564	11,996	**4.00**	0.00	**47,984**
	Recast	5424	6332	5424	12.45	2.86	67,503
	NEOGEN	**3,295**	**4,199**	**3,295**	16.13	6.33	53,142
	Grid	88,800	158,740	88,800	12.00	0.00	1,065,600
City	LCT	2556	2730	2556	9.00	0.00	23,004
	ECM	1442	1621	4679	14.42	2.16	67,491
	CDG	3449	5276	3449	**4.00**	0.00	**13,796**
	Recast	1645	1837	1645	11.88	3.13	19,548
	NEOGEN	**1,164**	**1,319**	**1,164**	13.61	5.17	15,846
	Grid	207,478	408,234	207,478	12.00	0.00	2,489,736

(*continued on next page*)

Table 7 (*continued*)

Environment	Navigation mesh	Complexity					
					Region complexity		
		$\|V\|$	$\|E\|$	$\|\mathcal{R}\|$	Average	SD	Total
Maze8	LCT	26	25	26	9.00	0.00	234
	ECM	30	29	51	14.71	2.32	750
	CDG	110	151	110	**4.00**	0.00	440
	Recast	15	14	15	11.20	1.33	**168**
	NEOGEN	**13**	**12**	**13**	14.77	2.99	192
	Grid	31	30	31	12.00	0.00	372
Maze16	LCT	82	81	82	9.00	0.00	738
	ECM	84	83	156	14.85	2.25	2,316
	CDG	566	950	566	**4.00**	0.00	2,264
	Recast	40	39	40	12.00	1.64	**480**
	NEOGEN	**39**	**38**	**39**	15.00	3.11	585
	Grid	127	126	127	12.00	0.00	1,524
Maze32	LCT	363	362	363	9.00	0.00	3,267
	ECM	358	357	686	14.89	2.23	10,212
	CDG	2255	3404	2255	**4.00**	0.00	9,020
	Recast	183	182	183	11.89	1.42	**2,175**
	NEOGEN	**175**	**174**	**175**	15.02	2.88	2,628
	Grid	511	510	511	12.00	0.00	6,132
Maze64	LCT	1421	1420	1421	9.00	0.00	12,789
	ECM	1392	1391	2672	14.88	2.24	39,756
	CDG	9783	14,721	9783	4.00	0.00	39,132
	Recast	722	721	722	11.88	1.56	**8,574**
	NEOGEN	**673**	**672**	**673**	15.09	3.00	10,155
	Grid	2047	2046	2047	12.00	0.00	24,564
Maze128	LCT	5681	5680	5681	9.00	0.00	51,129
	ECM	5567	5566	10,710	14.88	2.24	159,336
	CDG	25,135	44,910	25,135	4.00	0.00	100,540
	Recast	2862	2861	2862	11.95	1.59	**34,206**
	NEOGEN	**2,703**	**2,702**	**2,703**	15.08	2.98	40,755
	Grid	8191	8190	8191	12.00	0.00	98,292

Table 8
Results for the complexity metrics in the multi-layered environments. An empty row indicates that the navigation mesh could not be computed for the corresponding algorithm and environment. Numbers in boldface indicate the lowest metric value (among all methods) for a particular environment.

Environment	Navigation mesh	Complexity					
					Region complexity		
		$\|V\|$	$\|E\|$	$\|\mathcal{R}\|$	Average	SD	Total
as_oilrig	ECM	603	629	1283	14.73	2.23	18,894
	CDG	-	-	-	-	-	-
	Recast	**527**	**558**	**527**	12.01	3.04	**6,330**
	NEOGEN	-	-	-	-	-	-
	Grid	76,163	141,560	76,163	12.00	0.00	913,956
cs_assault	ECM	308	312	625	14.76	2.26	9,228
	CDG	2802	3212	2802	4.00	0.00	11,208
	Recast	334	363	334	10.98	2.53	3,666
	NEOGEN	**145**	**149**	**145**	15.35	5.27	**2,226**
	Grid	21,835	38,665	21,835	12.00	0.00	262,020
cs_siege	ECM	836	853	1712	14.46	2.26	24,756
	CDG	2482	2712	2482	4.00	0.00	9,928
	Recast	525	546	525	12.05	3.09	6,327
	NEOGEN	**362**	**375**	**362**	15.70	6.24	**5,682**
	Grid	26,278	47,145	26,278	12.00	0.00	315,336
de_dust	ECM	636	645	1271	14.16	2.14	18,003
	CDG	2272	2586	2272	4.00	0.00	9,088
	Recast	455	465	455	11.99	2.95	5,454
	NEOGEN	**302**	**309**	**302**	14.80	6.23	**4,470**
	Grid	25,170	44,065	25,170	12.00	0.00	302,040
de_dust2	ECM	552	556	1122	14.27	2.18	16,008
	CDG	2349	2904	2349	4.00	0.00	9,396
	Recast	403	406	403	11.69	3.10	4,713
	NEOGEN	**292**	**286**	**292**	13.87	5.27	**4,050**
	Grid	20,450	34,220	20,450	12.00	0.00	245,400

(*continued on next page*)

Table 8 (*continued*)

Environment	Navigation mesh	Complexity					
					Region complexity		
		$\|V\|$	$\|E\|$	$\|\mathcal{R}\|$	Average	SD	Total
Jardin	ECM	654	667	1395	14.51	2.43	20,238
	CDG	1753	2715	1753	4.00	0.00	7,012
	Recast	**212**	**231**	**212**	12.25	3.20	**2,598**
	NEOGEN	233	260	233	17.78	16.56	4,143
	Grid	2813	2842	2813	12.00	0.00	33,756
Neogen1	ECM	442	448	1152	14.66	2.39	16,890
	CDG	1766	2849	1766	4.00	0.00	7,064
	Recast	**150**	**159**	**150**	11.56	3.28	**1,734**
	NEOGEN	185	190	185	23.22	31.83	4,296
	Grid	4834	5843	4834	12.00	0.00	58,008
Neogen2	ECM	390	403	1238	14.87	2.30	18,414
	CDG	3880	5262	3880	4.00	0.00	15,520
	Recast	**249**	**264**	**249**	11.58	3.17	**2,883**
	NEOGEN	295	301	295	15.31	7.11	4,515
	Grid	9670	17,021	9670	12.00	0.00	116,040
Neogen3	ECM	439	439	985	14.55	2.34	14,328
	CDG	3571	5834	3571	4.00	0.00	14,284
	Recast	262	263	262	11.83	3.20	**3,099**
	NEOGEN	**217**	**217**	**217**	14.90	8.27	3,234
	Grid	9781	17,854	9781	12.00	0.00	117,372
Dungeon	ECM	633	625	1370	14.92	2.43	20,445
	CDG	1317	1546	1317	4.00	0.00	5,268
	Recast	**256**	**260**	**256**	12.22	3.08	**3,129**
	NEOGEN	-	-	-	-	-	-
	Grid	2131	2918	2131	12.00	0.00	25,572
NavTest	ECM	365	362	812	14.87	2.43	12,078
	CDG	2665	3335	2665	4.00	0.00	10,660
	Recast	182	179	182	12.25	3.06	2,229
	NEOGEN	**98**	**109**	**98**	18.46	10.78	**1,809**
	Grid	5280	6661	5280	12.00	0.00	63,360
ParkingLot	ECM	61	68	108	15.08	2.33	1,629
	CDG	803	1399	803	4.00	0.00	3,212
	Recast	**50**	**57**	**50**	11.88	2.40	**594**
	NEOGEN	-	-	-	-	-	-
	Grid	2094	3781	2094	12.00	0.00	25,128
Library	ECM	216	218	377	14.50	2.36	5,466
	CDG	2866	3873	2866	4.00	0.00	11,464
	Recast	122	124	122	11.95	2.66	**1,458**
	NEOGEN	**74**	**70**	**74**	20.64	8.85	1,527
	Grid	3188	5670	3188	12.00	0.00	38,256
Tower	ECM	4988	5019	9421	14.37	2.38	135,345
	CDG	17,970	27,866	17,970	4.00	0.00	71,880
	Recast	**1,467**	**1,701**	**1,467**	12.07	2.87	**17,709**
	NEOGEN	-	-	-	-	-	-
	Grid	12,442	22,074	12,442	12.00	0.00	149,304
BigCity	ECM	32,175	32,562	69,181	14.41	2.37	997,236
	CDG	-	-	-	-	-	-
	Recast	**9,380**	**11,326**	**9,380**	12.27	3.11	**115,137**
	NEOGEN	-	-	-	-	-	-
	Grid	-	-	-	-	-	-

As expected, exact methods scaled better to large environments than some voxel-based methods: while the LCT and the ECM remained fast, the running times increased strongly for Recast and the CDG in particular. NEOGEN was usually the fastest voxel-based method, remaining on par with the ECM in most environments. The *BigCity* environment challenged the limits of all voxel-based methods: only Recast could produce a result (when using a larger voxel size), while all other programs crashed, even with coarser sampling settings. Decreasing the voxel size caused Recast to crash as well, most likely due to memory usage. Recast can subdivide the environment into *tiles* to alleviate this, but we have excluded this option to simplify our comparison.

These differences in scalability are difficult to judge because voxel-based methods include the reconstruction of $\mathcal{E}_{free}$ in their algorithm. Combined with the results for coverage and connectivity, this indicates that obtaining $\mathcal{E}_{free}$ *without* voxels is an interesting topic for future work. NEOGEN already does this to some extent (i.e. it does not use voxels for everything), which already appears to make NEOGEN more scalable than the CDG or Recast.

8. Conclusions

A navigation mesh enables path planning and crowd simulation for walking characters in 2D and 3D environments. In this paper,

Table 9
Results for the performance metrics in the 2D environments. The descriptions of all metrics can be found in Section 6. Numbers in boldface indicate the lowest metric value (among all methods) for a particular environment.

Environment	Navigation mesh	Performance			
		Construction time (ms)		Memory usage (MB)	
		Average	SD	Average	SD
Simple	LCT	**1.20**	0.28	**1.13**	0.02
	ECM	5.97	0.82	33.45	0.05
	CDG	12,597.63	164.29	57.07	0.09
	Recast	375.19	19.96	20.83	0.14
	NEOGEN	7.80	1.08	67.97	0.18
	Grid	94.49	10.13	27.02	0.08
Military	LCT	**1.10**	0.15	**1.14**	0.02
	ECM	7.85	1.12	33.46	0.12
	CDG	28,310.23	654.68	64.63	0.17
	Recast	7,796.13	36.15	92.37	0.15
	NEOGEN	12.45	2.82	69.36	0.24
	Grid	278.86	29.92	62.37	2.22
University	LCT	**8.93**	0.88	**1.35**	0.11
	ECM	35.35	3.61	36.84	0.06
	CDG	15,160.81	160.35	50.49	0.14
	Recast	614.44	15.37	24.92	0.22
	NEOGEN	48.05	3.58	79.21	0.59
	Grid	115.21	12.81	32.16	0.12
Zelda	LCT	**5.12**	0.59	**1.23**	0.02
	ECM	26.27	2.05	36.14	0.07
	CDG	14,433.28	101.29	53.24	0.10
	Recast	297.77	13.71	19.67	0.17
	NEOGEN	29.75	1.55	74.98	0.30
	Grid	94.37	7.32	29.04	0.05
Zelda2x2	LCT	**23.23**	2.16	**2.03**	0.03
	ECM	112.19	9.12	46.32	0.11
	CDG	18,477.72	155.17	55.99	0.20
	Recast	1,521.38	33.44	74.46	0.49
	NEOGEN	124.55	9.30	99.87	0.29
	Grid	201.03	12.79	49.18	0.51
Zelda4x4	LCT	**93.98**	6.26	**4.98**	0.03
	ECM	442.78	21.11	84.74	0.39
	CDG	33,448.68	270.42	65.71	0.21
	Recast	6,310.88	53.18	291.51	0.23
	NEOGEN	516.55	15.83	197.73	0.12
	Grid	947.03	13.08	121.54	3.01
City	LCT	**40.94**	2.28	**2.30**	0.04
	ECM	167.30	9.50	49.78	0.15
	CDG	29,421.39	277.12	64.31	0.12
	Recast	9,833.90	76.47	134.09	0.01
	NEOGEN	180.90	13.33	104.41	0.18
	Grid	2,419.58	112.02	191.66	12.81
Maze8	LCT	**0.24**	0.05	**1.10**	0.02
	ECM	1.45	0.18	33.00	0.01
	CDG	269.66	6.11	21.77	0.17
	Recast	1.40	0.48	1.91	0.03
	NEOGEN	1.60	0.73	67.53	0.22
	Grid	63.91	6.40	19.84	0.06
Maze16	LCT	**0.76**	0.14	**1.13**	0.02
	ECM	4.14	0.35	33.36	0.07
	CDG	665.82	9.56	23.61	0.20
	Recast	10.27	1.34	2.28	0.07
	NEOGEN	5.05	0.92	68.16	0.19
	Grid	63.50	3.69	21.07	0.56
Maze32	LCT	**4.17**	0.46	**1.19**	0.00
	ECM	17.79	0.68	35.50	0.04
	CDG	3,162.39	124.76	33.14	0.21
	Recast	68.02	7.51	4.03	0.08
	NEOGEN	21.95	1.43	75.13	0.39
	Grid	66.49	3.47	23.79	0.04

(*continued on next page*)

Table 9 (*continued*)

Environment	Navigation mesh	Performance			
		Construction time (ms)		Memory usage (MB)	
		Average	SD	Average	SD
Maze64	LCT	**14.53**	1.76	**1.79**	0.02
	ECM	71.13	4.00	44.15	0.19
	CDG	23,876.64	168.19	55.87	0.21
	Recast	273.41	14.18	9.66	0.30
	NEOGEN	85.45	4.65	93.41	0.18
	Grid	93.23	7.98	30.40	0.06
Maze128	LCT	**57.57**	4.03	**4.22**	0.04
	ECM	309.60	11.58	77.14	0.34
	CDG	61,867.47	349.41	67.76	0.20
	Recast	1,115.27	32.11	33.67	0.54
	NEOGEN	350.60	11.77	171.75	0.22
	Grid	204.91	5.83	63.67	0.17

Table 10
Results for the performance metrics in the multi-layered environments. An empty row indicates that the navigation mesh could not be computed for the corresponding algorithm and environment. Numbers in boldface indicate the lowest metric value (among all methods) for a particular environment.

Environment	Navigation mesh	Performance			
		Construction time (ms)		Memory usage (MB)	
		Average	SD	Average	SD
as_oilrig	ECM	**58.14**	1.74	**40.56**	0.17
	CDG	-	-	-	-
	Recast	11,549.74	38.86	178.34	0.33
	NEOGEN	-	-	-	-
	Grid	2,627.48	22.19	686.75	0.09
cs_assault	ECM	**16.04**	0.43	**36.90**	0.20
	CDG	11,762.23	120.80	51.30	0.20
	Recast	2,650.00	23.16	63.54	0.01
	NEOGEN	22.20	1.21	72.75	0.32
	Grid	689.74	15.69	194.07	0.29
cs_siege	ECM	56.34	0.95	**41.60**	0.19
	CDG	8,533.73	108.86	45.21	0.16
	Recast	3,200.50	22.28	105.15	0.01
	NEOGEN	**55.85**	1.98	81.88	0.18
	Grid	1,399.37	25.35	483.14	0.09
de_dust	ECM	60.36	0.78	**39.38**	0.17
	CDG	7,226.04	70.76	44.23	0.17
	Recast	3,037.14	25.38	101.23	0.02
	NEOGEN	**46.00**	2.45	79.81	0.31
	Grid	583.62	11.13	174.64	0.08
de_dust2	ECM	65.72	0.92	**38.11**	0.10
	CDG	10,121.20	118.74	50.90	0.22
	Recast	2,265.68	27.08	67.13	0.01
	NEOGEN	**40.65**	2.92	76.21	0.19
	Grid	492.85	13.57	137.71	0.53
Jardin	ECM	**115.60**	2.13	41.09	0.14
	CDG	34,541.99	250.60	78.77	0.23
	Recast	181.04	5.89	**9.42**	0.05
	NEOGEN	486.75	9.60	107.22	0.28
	Grid	370.58	6.66	75.33	2.19
Neogen1	ECM	**162.66**	6.25	49.52	0.11
	CDG	41,102.63	239.13	89.02	0.21
	Recast	816.09	16.65	**14.10**	0.00
	NEOGEN	1,613.20	41.21	136.10	0.17
	Grid	604.13	24.79	136.35	14.39
Neogen2	ECM	**52.58**	1.59	39.35	0.16
	CDG	92,932.59	608.03	145.79	0.28
	Recast	798.30	14.23	**23.50**	0.20
	NEOGEN	183.35	7.44	87.09	0.24
	Grid	361.75	9.71	58.12	5.02

(*continued on next page*)

Table 10 (*continued*)

Environment	Navigation mesh	Performance			
		Construction time (ms)		Memory usage (MB)	
		Average	SD	Average	SD
Neogen3	ECM	**26.83**	0.74	38.41	0.16
	CDG	24,093.69	275.04	59.29	0.18
	Recast	907.61	16.33	**23.37**	0.11
	NEOGEN	59.85	3.48	76.08	0.43
	Grid	481.46	10.69	86.99	1.32
Dungeon	ECM	**68.12**	0.94	39.99	0.20
	CDG	5,284.11	43.13	38.95	0.16
	Recast	143.82	5.69	**10.92**	0.01
	NEOGEN	-	-	-	-
	Grid	428.65	10.88	71.02	0.87
NavTest	ECM	**39.67**	0.58	37.87	0.17
	CDG	15,004.41	135.66	56.44	0.24
	Recast	367.19	23.71	**16.56**	0.04
	NEOGEN	46.40	1.62	74.16	0.41
	Grid	260.76	6.60	49.62	0.53
ParkingLot	ECM	**8.89**	0.33	34.27	0.11
	CDG	14,380.66	88.70	61.95	0.12
	Recast	88.87	6.92	**6.32**	0.01
	NEOGEN	-	-	-	-
	Grid	119.22	2.16	25.21	0.06
Library	ECM	**15.04**	0.46	35.95	0.14
	CDG	20,841.33	110.17	67.26	0.11
	Recast	163.97	5.52	**10.27**	0.17
	NEOGEN	23.00	1.79	71.86	0.19
	Grid	211.74	10.35	30.60	0.09
Tower	ECM	**143.20**	7.35	75.30	0.46
	CDG	242,254.50	4.50	106.59	145.06
	Recast	1,281.85	26.13	**33.29**	0.00
	NEOGEN	-	-	-	-
	Grid	1,323.31	13.15	174.45	4.69
BigCity	ECM	**893.74**	30.44	308.02	0.59
	CDG	-	-	-	-
	Recast	12,972.87	79.74	**191.40**	0.01
	NEOGEN	-	-	-	-
	Grid	-	-	-	-

we have performed a comparative study of multiple state-of-the-art navigation meshes. We have proposed properties by which a mesh and its construction algorithm can be classified, and metrics that objectively measure the quality of a mesh in practice. We have used these components to compare the Local Clearance Triangulation, the Explicit Corridor Map, the Clearance Disk Graph, NEOGEN, Recast Navigation, and a grid.

Although the study can be extended in various ways, our results already suggest interesting properties. Voxel-based methods can yield compact navigation meshes and even good coverage, but their algorithms does not seem to scale well to physically large environments. Furthermore, grids are usually not space-efficient representations of $\mathcal{E}_{free}$, although they may be attractive for particular applications, and we have not yet explored the possibilities of adaptive grid resolutions. Finally, exact methods often yield more complex navigation meshes because they capture all details. These methods could benefit from simplifications of $\mathcal{E}_{free}$.

The goal of this paper was not to find 'the best' navigation mesh, but to develop a way of comparing them via theoretical properties and quantitative metrics. We expect that our study will set a standard for the evaluation and develop-ment of navigation meshes, and that it can help users choose an appropriate navigation mesh for their applications.

9. Future work

Based on the findings of this paper, we identify several possible directions for future work.

9.1. Environment dataset

A limitation of our comparison lies in the current set of input environments. We have focused on examples from previous publications and from an existing popular 3D game. Also, we have deliberately only used 'clean' walkable environments and not raw 3D geometry, to allow a fair comparison between exact and voxel-based methods.

We hope that our set of 2D and 3D input environments can be the start of an open dataset for navigation-mesh research, similarly to the ones that exist for local steering [37] and grid-based search [34,36]. Finding a set of environments that covers the complete 'problem space' (i.e. that contains all types of features that can occur in practice) is a challenging direction for future work.

9.2. Variants of grids

As expected, our simple fixed-resolution grid turned out to be an inefficient navigation mesh for many environments. This navigation mesh can be improved in various ways [52,53] so that its cell size adapts locally to the required level fo detail. This will make the grid more suitable for large or detailed input. For a more comprehensive overview of all possible navigation meshes, it would be interesting to include such 'advanced grids' in a future comparison.

9.3. Parameter settings

To simplify the comparison, we have chosen a single set of parameter settings for all methods. However, for navigation meshes such as Recast, (manual) parameter tuning is considered to be part of the navigation-mesh construction process. For any given environment, it is likely that there are specific parameter settings that yield better results. It would therefore be interesting to analyse further how parameter settings affect the output. For example, Oliva and Pelechano have discussed how the voxel size affects the results of Recast and NEOGEN [10]. In future work, we could possibly use the metrics from this paper to automatically find the best parameter settings for a specific environment.

9.4. Path-related metrics

We would also like to investigate metrics that are more related to paths than to the navigation meshes themselves. For instance, it would be useful to measure the efficiency of a navigation mesh for path planning: how much time does it take to compute paths in $\mathcal{G}$, and how efficiently can these be converted to geometric routes using the regions of $\mathcal{R}$? We have currently excluded such metrics because they are heavily implementation-dependent; each navigation mesh implementation can apply its own improvements that make a comparison difficult or unfair. A generic implementation of path planning for embedded graphs would be more objective, but it would also miss the potential advantages of each specific navigation mesh. For now, we are confident that our complexity metrics are good indicators of path planning efficiency.

To go even further, another option is to look at the *quality* of the routes that are computed: how short are they, and how well do they correspond to real-life behavior? We have excluded this aspect because route quality is not necessarily a property of the navigation mesh itself, but more of the (path planning and smoothing) algorithms applied to it. Ultimately, we would like to quantify how well a navigation mesh captures the navigation abilities of real humans. This is a challenging research question that is still largely unanswered. We expect that not everything can be analyzed mechanically, and that user studies will also be required.

It is worth noting that there have recently been developments in path-planning research that reduce the dependency on the specific navigation mesh that is used. For example, there are now *any-angle path-planning* algorithms that can compute shortest paths from arbitrary polygonal navigation meshes [57]. Using such algorithms, different navigation meshes can (in theory) be used to compute the exact same paths. Furthermore, there are *hierarchical* algorithms that speed up path-planning queries by exploiting the hierarchy of any navigation mesh [58]. These developments suggest that path planning and navigation-mesh quality are turning more and more into disjoint topics that can (and should) be treated separately.

9.5. 3D-Environment processing

Finally, an important topic for future work is the development of robust and scalable algorithms that convert a 'raw' 3D environment to a 'clean' walkable environment. Our experiments suggest that voxel-based approaches are not scalable to large environments; however, they provide the advantage of simplifying the input environment, leading to possibly simpler navigation meshes. *Exact* filtering algorithms (that do not use voxels) should scale better, but they may be sensitive to small details or imperfections in the input (such as gaps or overlap), and they may be more likely to crash if the geometry contains difficult exceptional cases.

Since our conference publication in 2016 [5], there has been some work on environment pre-processing using aspects of computational geometry [17,19] and graph theory [39]. However, some of the corresponding software relies on exact number types for guaranteed robustness, which greatly affects its efficiency. Some geometric algorithms, such as the construction algorithm of the LCT [14], can be implemented with non-exact numbers by leveraging robust geometric predicates [59]. Unfortunately, this does not apply for the conversion of a 3DE to a WE because it is treated a 'pipeline' of steps [17], where the output of each step needs to be exact to guarantee correctness. To make the conversion pipeline robust *and* computationally efficient, future work should focus on relaxing this number-type constraint. In the end, the best solution might be to combine various approaches, such as a filtering pipeline where each step is based on rounded coordinates.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Declaration of Competing Interest

None.

CRediT authorship contribution statement

Wouter van Toll: Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Roy Triesscheijn:** Conceptualization, Methodology, Software, Validation, Investigation, Writing - original draft, Visualization. **Marcelo Kallmann:** Conceptualization, Methodology, Validation, Resources, Writing - review & editing. **Ramon Oliva:** Conceptualization, Methodology, Validation, Resources. **Nuria Pelechano:** Conceptualization, Methodology, Validation, Writing - review & editing. **Julien Pettré:** Conceptualization, Methodology, Validation, Resources, Writing - review & editing, Supervision, Project administration, Funding acquisition. **Roland Geraerts:** Conceptualization, Methodology, Validation, Resources, Writing - review & editing, Supervision, Project administration, Funding acquisition.

Acknowledgments

We thank all research groups involved in this study for sharing their source code and environments, and for joining us in helpful discussions. Furthermore, we thank Mihai Polak and Arne Hillebrand for helping us with pre-processing 3D geometry using their software.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cag.2020.06.006.

References

[1] van Toll W, Jaklin N, Geraerts R. Towards believable crowds: A generic multi--level framework for agent navigation. ASCI.OPEN; 2015.
[2] Lozano-Perez T. Spatial planning: a configuration space approach. IEEE Trans Computing 1983;32(2):108–20.
[3] Kavraki LE, Švestka P, Latombe J-C, Overmars MH. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans Robot Autom 1996;12(4):566–80.
[4] LaValle SM. Planning algorithms. Cambridge University Press; 2006.
[5] van Toll W, Triesscheijn R, Kallmann M, Oliva R, Pelechano N, Pettré J, et al. A comparative study of navigation meshes. In: Proc. 9th ACM SIGGRAPH Int. Conf. Motion in Games; 2016. p. 91–100.
[6] Snook G. Simplified 3D movement and pathfinding using navigation meshes. In: DeLoura M, editor. Game Programming Gems. Charles River Media; 2000. p. 288–304.

[7] Tozour P. Building a near-optimal navigation mesh. In: Rabin S, editor. AI Game Programming Wisdom. Charles River Media; 2002. p. 171–85.
[8] Deusdado L, Fernandes AR, Belo O. Path planning for complex 3D multilevel environments. In: Proc. 24th Spring Conf. Computer Graphics; 2008. p. 187–94.
[9] Mononen M.. Recast Navigation. https://github.com/recastnavigation/recastnavigation/; 2019.
[10] Oliva R, Pelechano N. NEOGEN: Near optimal generator of navigation meshes for 3D multi-layered environments. Computers & Graphics 2013;37(5):403–12.
[11] Pettré J, Laumond J-P, Thalmann D. A navigation graph for real-time crowd animation on multilayered and uneven terrain. In: Proc. 1st Int. Workshop on Crowd Simulation; 2005. p. 81–9.
[12] Geraerts R. Planning short paths with clearance using Explicit Corridors. In: Proc. IEEE Int. Conf. Robotics and Automation; 2010. p. 1997–2004.
[13] Hale DH, Youngblood GM, Dixit PN. Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds. In: Proc. 4th Artificial Intelligence and Interactive Digital Entertainment Conf.; 2008. p. 173–8.
[14] Kallmann M. Dynamic and robust Local Clearance Triangulations. ACM Trans Graph 2014;33(5).
[15] Oliva R, Pelechano N. Automatic generation of suboptimal navmeshes. In: Proc. 4th Int. Conf. Motion in Games; 2011. p. 328–39.
[16] van Toll WG, Cook IV AF, Geraerts R. Navigation meshes for realistic multi--layered environments. In: Proc. 24th IEEE/RSJ Int. Conf. Intelligent Robots and Systems; 2011. p. 3526–32.
[17] Polak RM. Extracting walkable areas from 3D environments. Utrecht University; 2016. Master's thesis.
[18] Oliva R. A framework for navigation of autonomous characters in complex virtual environments. Universitat Politècnica de Catalunya; 2016.
[19] Vermeulen JL, Hillebrand A, Geraerts R. Annotating traversable gaps in walkable environments. In: Proc. IEEE Int. Conf. Robotics and Automation; 2018. p. 3045–52.
[20] Hale DH, Youngblood GM. Full 3D spacial decomposition for the generation of navigation meshes. In: Proc. 5th Artificial Intelligence and Interactive Digital Entertainment Conf.; 2009. p. 143–7.
[21] Lamarche F. TopoPlan: a topological path planner for real time human navigation under floor and ceiling constraints. Comput Graph Forum 2009;28(2):649–58.
[22] Ricks BC, Egbert PK. A whole surface approach to crowd simulation on arbitrary topologies. IEEE Trans Vis Comput Graphics 2014;20:159–71.
[23] Berseth G, Kapadia M, Faloutsos P. ACCLMesh: Curvature-based navigation mesh generation. In: Proc. 8th ACM SIGGRAPH Conf. Motion in Games; 2015. p. 97–102.
[24] Lopez T, Lamarche F, Li T-Y. Space-time planning in changing environments: using dynamic objects for accessibility. Comput Animat Virtual Worlds 2012;23:87–99.
[25] Budde S. Automatic generation of jump links in arbitrary 3D environments for navigation meshes. Humboldt-Universität zu Berlin; 2013. Master's thesis.
[26] Kapadia M, Xianghao X, Nitti M, Kallmann M, Coros S, Sumner R. PRECISION: Precomputing environment semantics for contact-rich character animation. In: Proc. 20th ACM SIGGRAPH Symp. Interactive 3D Graphics and Games; 2016. p. 29–37.
[27] Jaklin NS, Cook IV AF, Geraerts R. Real-time path planning in heterogeneous environments. Comput Animat Virtual Worlds 2013;24(3):285–95.
[28] Jaklin NS, Tibboel M, Geraerts R. Computing high-quality paths in weighted regions. In: Proc. 7th Int. Conf. Motion in Games; 2014. p. 77–86.
[29] Ali S, Nishino K, Manocha D, Shah M. Modeling, simulation and visual analysis of crowds: a multidisciplinary perspective. Springer; 2013.
[30] Kapadia M, Pelechano N, Allbeck J, Badler NI. Virtual crowds: steps toward behavioral realism. Morgan & Claypool Publishers; 2015.
[31] Pelechano N, Allbeck JM, Kapadia M, Badler NI. Simulating heterogeneous crowds with interactive behaviors. CRC Press; 2016.
[32] Thalmann D, Musse SR. Crowd simulation. 2. Springer; 2013.
[33] Curtis S, Best A, Manocha D. Menge: a modular framework for simulating crowd movement. Collective Dynamics 2016;1(A1):1–40.
[34] Sturtevant N. Benchmarks for grid-based pathfinding. Trans Computational Intelligence and AI in Games 2012;4(2):144–8. http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf
[35] Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Systems Science and Cybernetics 1968;4(2):100–7.
[36] Brewer D, Sturtevant NR. Benchmarks for pathfinding in 3D voxel space. In: Proc. Symposium on Combinatorial Search; 2018.
[37] Singh S, Kapadia M, Faloutsos P, Reinman G. An open framework for developing, evaluating, and sharing steering algorithms. In: Proc. 2nd Int. Workshop on Motion in Games; 2009. p. 158–69.
[38] Reitsma PSA, Pollard NS. Evaluating motion graphs for character animation. ACM Trans Graph 2007;26(4). 18es
[39] Hillebrand A, van den Akker JM, Geraerts R, Hoogeveen JA. Performing multicut on walkable environments. In: Proc. 10th Int. Conf. Combinatorial Optimization and Applications; 2016. p. 311–25.
[40] Hillebrand A, van den Akker JM, Geraerts R, Hoogeveen JA. Separating a walkable environment into layers. In: Proc. 9th ACM SIGGRAPH Int. Conf. Motion in Games; 2016. p. 101–6.
[41] van Toll W, Cook IV AF, van Kreveld MJ, Geraerts R. The medial axis of a multi-layered environment and its application as a navigation mesh. ACM Trans Spatial Algorithms and Systems 2018;4(1) 2:1-2:34.
[42] Unity3D Game Engine. http://www.unity3d.com/; 2016.
[43] Oliva R, Pelechano N. A generalized exact arbitrary clearance technique for navigation meshes. In: Proc. 6th Int. Conf. Motion in Games; 2013. p. 103–10.
[44] García FM, Kapadia M, Badler NM. GPU-based dynamic search on adaptive resolution grids. In: Proc. IEEE Int. Conf. Robotics and Automation; 2014. p. 1631–8.
[45] Harabor D, Grastien A. Online graph pruning for pathfinding on grid maps. In: Proc. 52th AAAI Conf. Artificial Intelligence; 2011. p. 1114–19.
[46] Lee W, Lawrence R. Fast grid-based path finding for video games. In: Advances in Artificial Intelligence. In: Lecture Notes in Computer Science, 7884. Springer; 2013. p. 100–11.
[47] Sturtevant N, Rabin S. Canonical orderings on grids. In: Proc. Int. Joint Conf. Artificial Intelligence; 2016. p. 683–9.
[48] Botea A, Müller M, Schaeffer J. Near optimal hierarchical path-finding. Journal of Game Development 2004;1:7–28.
[49] Koenig S, Likhachev M. D* Lite. In: Proc. AAAI Conf. of Artificial Intelligence; 2002. p. 476–83.
[50] Koenig S, Likhachev M, Furcy D. Lifelong Planning A*. Artif Intell 2004;155(1–2):93–146.
[51] Likhachev M, Ferguson D, Gordon G, Stentz A, Thrun S. Anytime Dynamic A*: An anytime, replanning algorithm. In: Proc. Int. Conf. Automated Planning and Scheduling; 2005. p. 262–71.
[52] Finkel RA, Bentley JL. Quad trees: a data structure for retrieval on composite keys. Acta Informatica 1974;4:1–9.
[53] Sturtevant N. A sparse grid representation for dynamic three-dimensional worlds. In: Proc. AAAI Conf. Artificial Intelligence and Interactive Digital Entertainment; 2011.
[54] Boost. The Boost C++ library. http://www.boost.org/; 2019.
[55] Valve Software. Counter-Strike 1.6. https://www.valvesoftware.com/; 2003.
[56] cs-bg. Fan-made Counter-Strike 1.6 map repository. http://maps.cs-bg.info/maps/cs/; accessed in 2017.
[57] Cui ML, Harabor DH, Grastien A. Compromise-free pathfinding on a navigation mesh. In: Proc. 26th Int. Joint Conf. Artificial Intelligence; 2017. p. 496–502.
[58] Rahmani V, Pelechano N. Multi-agent parallel hierarchical path finding in navigation meshes (MA-HNA*). Computers & Graphics 2020;86:1–14.
[59] Shewchuk JR. Robust adaptive floating-point geometric predicates. In: Proc. 12th Symp. Computational Geometry; 1996. p. 141–50.