# A Comparative Study of Navigation Meshes

Wouter van Toll[*] Utrecht University
Roy Triesscheijn Utrecht University
Marcelo Kallmann University of California, Merced
Ramon Oliva Universitat Politecnica de Catalunya
Nuria Pelechano Universitat Politecnica de Catalunya
Julien Pettré INRIA
Roland Geraerts Utrecht University

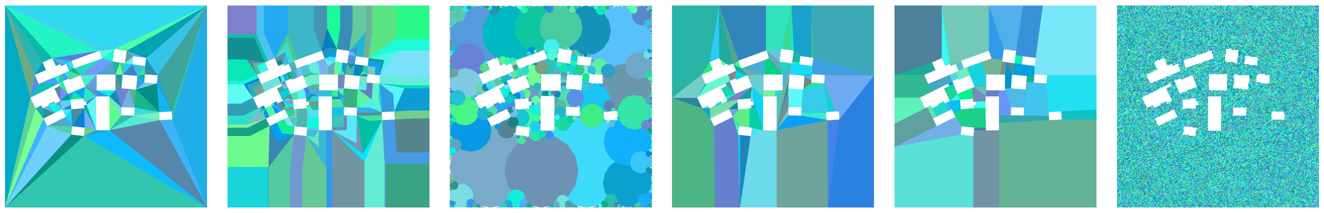


**Figure 1:** *Navigation meshes computed for the* Military *environment. Regions are shown in different colors. From left to right: the Local Clearance Triangulation, the Explicit Corridor Map, the Clearance Disk Graph, Recast, NEOGEN, and a grid.*

## 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 conduct the first comparative study of 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. Finally, we use these metrics to compare various state-of-the-art navigation meshes in a range of 2D and 3D environments.

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 on navigation meshes in interesting directions.

**Keywords:** navigation meshes, path planning, comparative study

**Concepts:** •**Computing methodologies** → **Mesh geometry models;** *Motion path planning;* •**General and reference** → **Metrics;** *Evaluation;*

[*]Corrected author. E-mail: W.G.vanToll@uu.nl.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org. © 2016 ACM.

*MiG '16*, October 10-12, 2016, Burlingame, CA, USA
ISBN: 978-1-4503-4592-7/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2994258.2994262

## 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 [van Toll et al. 2015]. The 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 [Lozano-Perez 1983] is often represented as a sampling-based graph (e.g. [Kavraki et al. 1996; LaValle 2006]). 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 an optimal 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. 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. This difference makes the two categories difficult to compare. Furthermore, each method uses its own set of test environments to show its (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 types of 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.

The main contributions of this paper are the following:

- We propose properties by which the data structures and algorithms of navigation meshes can be classified (Section 4).
- We present quantitative metrics to measure the quality and performance of a navigation mesh implementation for a given input environment (Section 5). 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 6).

We emphasize that our goal is not to expose which navigation mesh implementation works best for particular input environments. Instead, by comparing navigation meshes using a common test platform and settings, we intend to set a standard for the analysis of navigation meshes and to expose interesting areas for future research.

## 2 Related Work

### 2.1 Navigation Meshes

Snook [2000] and Tozour [2002] 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 [Deusdado et al. 2008; Mononen 2014; Oliva and Pelechano 2013b; Pettré et al. 2005] 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 the free space is extracted from the 3D geometry.

*Exact* methods [Geraerts 2010; Hale et al. 2008; Kallmann 2014; Oliva and Pelechano 2011; van Toll et al. 2011; Tozour 2002] 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 [Polak 2016].

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 [Hale and Youngblood 2009; Lamarche 2009]. Alternatively, one could perform crowd simulations on arbitrary surfaces with no consistent direction of gravity [Ricks and Egbert 2014; Berseth et al. 2015]. Other methods allow characters to jump between surfaces by either checking for jumping possibilities on the fly [Lopez et al. 2012] or annotating a navigation mesh with jump links beforehand [Budde 2013]. However, these extensions are outside the scope of our comparative 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 [Ali et al. 2013; Kapadia et al. 2015; Pelechano et al. 2016; Thalmann and Musse 2013]. Also, there are multiple crowd simulation frameworks in which navigation meshes play a central role [Curtis et al. 2014; van Toll et al. 2015]. 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. However, crowd simulation is an important motivation for many of the properties and metrics that we will propose.

### 2.2 Comparative Studies

Our comparative study of navigation meshes is inspired by comparisons for *other* aspects of path planning and crowd simulation.

Sturtevant [2012] has developed a test set of environments for 2D *grid-based* path planning. This set includes mazes of various complexities and levels from computer games, and it is often used to analyze variants of the A* search algorithm [Hart et al. 1968]. Although we study general navigation meshes rather than grids, we will also include grid-based environments in our experiments.

SteerBench [Singh et al. 2009] 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.

In this paper, we compare navigation meshes in a similar way based on metrics, input environments, and a single test platform. We will see that parameter settings are influential in our study as well.

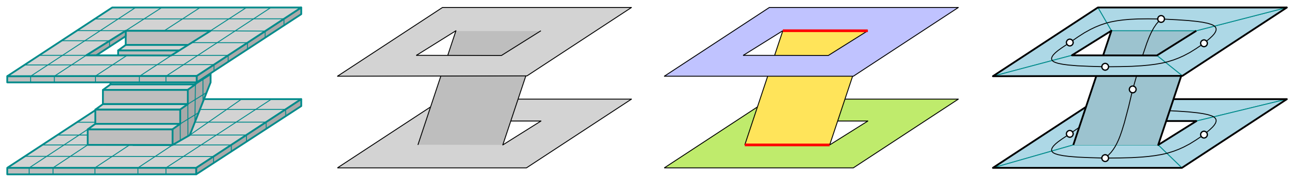
## 3 Definitions

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 5 requires unified definitions.

### 3.1 2D Environment

A *2D environment* 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 2D environments in $\mathbb{R}^3$ by assigning a height component of zero to each vertex.



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

**Figure 2:** *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.* ***(d)*** *A navigation mesh is a description of a WE for path planning purposes.*

## 3.2 3D Environment

In this paper, a *3D environment* (3DE) is a raw collection of polygons in $\mathbb{R}^3$. These polygons may include floors, ceilings, walls, or any other type of geometry. Figure 2a shows an example. To define the *free space* $\mathcal{E}_{free}$ of a 3DE, we need to specify various 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 vertical distance between a floor and a ceiling.

Characters are typically approximated by cylinders. Some navigation meshes use a predefined character radius to determine $\mathcal{E}_{free}$. In this paper, we will use a radius of zero to enable an objective comparison to other navigation meshes.

## 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. Figure 2b shows an example. In our experiments, all environments will be WEs, so we know beforehand which area should be covered by a navigation mesh.

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. The free space $\mathcal{E}_{free}$ is simply the set of polygons itself. 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 2D environments 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) [Pettré et al. 2005; van Toll et al. 2011] 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 Figure 2c.

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, but good results can be obtained using heuristics [Hillebrand 2012]. In our experiments, we will subdivide all WEs into layers to facilitate the construction of navigation meshes.

Finally, note that a 2D environment is a special case of an MLE with only one layer (or, equivalently, a WE that can be projected onto $P$ without overlap). Section 5 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 $\mathcal{R}$.

Figure 2d shows an abstract example of a navigation mesh. For many navigation meshes, $\mathcal{R}$ consists of non-overlapping simple polygons, and $\mathcal{G}$ is 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 [Pettré et al. 2005], $\mathcal{R}$ consists of overlapping disks, and $\mathcal{G}$ contains an edge wherever two disks overlap. The Explicit Corridor Map [van Toll et al. 2011] is explicitly defined as a graph, and the mesh regions can be derived from its annotations. Still, all meshes have in common that $\mathcal{R}$ and $\mathcal{G}$ can be obtained from their representation in some way.

# 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.

**Region type** The type of regions in $\mathcal{R}$, e.g. triangles 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 or not the regions in $\mathcal{R}$ can overlap by definition. Having overlapping regions is generally discouraged because geometric algorithms that assume non-overlapping regions may not work properly. Also, a query point (or an agent) can be in multiple regions at the same time in case of overlap, which may complicate path planning and crowd simulation.

**Pipeline** The conversion pipeline 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.

**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 or not 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 or not the navigation mesh supports dynamic insertions and deletions of obstacles.

Due to space constraints, we cannot include our full theoretical comparison of the navigation meshes that will be used in our experiments. Instead, Section 6.1 will describe each method briefly, and Table 1 classifies each method by the properties described above. We will include the full comparison in an extended publication.

## 5 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:

1. (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.

2. (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.

3. (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.

4. (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.

Of course, many other questions are interesting, e.g. questions related to the peformance of path planning queries, or to the quality or realism of paths. We will discuss a number of options in Section 7 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 properties can only be analyzed 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.

### 5.1 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 are based on the assumption that the environment has a consistent direction of gravity. Coverage is the only category of metrics in which this assumption comes into play.

#### 5.1.1 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 straightforward 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, there should be 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 (and if not, $p$ is assumed to lie in $\mathcal{E}_{obs}$). Several choices can be made here, such as the maximum allowed height difference between $p$ and $m(p)$. We will describe our own implementation of $m$ in Section 6.

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. $R_i^* = \{m(p) \mid p \in R_i \text{ and } m(p) \text{ exists}\}$. Because each mapped region is a subset of $\mathcal{E}_{free}$, we can use the mapped regions to define unions, coverage, and overlap. 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.

#### 5.1.2 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.

### 5.1.3 Coverage Metrics

We introduce three coverage metrics. Each metric has a regular version $M$ with range $\mathbb{R}_{\geq 0}$ 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 $\mathbb{R}^3$. High coverage is desirable: it allows characters to use more of $\mathcal{E}_{free}$.

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

**Incorrect area** 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.

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

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

**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.

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

If a navigation mesh is deliberately based on overlapping regions (e.g. [Pettré et al. 2005]), then this metric simply indicates how much space is covered more than once. Otherwise, overlap may indicate an implementation bug, which is not the focus of our comparative study.

## 5.2 Connectivity

The second set of metrics analyzes how well the graph $\mathcal{G} = (V, E)$ represents the dual graph of $\mathcal{E}_{free}$.

**# 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 when 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 is perfect, the geometry of $\mathcal{R}$ is not necessarily correct.

## 5.3 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, and the complexity of these regions may have implications for 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 the simulation may move from one region to another. Moving to another region typically triggers computational overhead in the simulation. Hence, having fewer regions may cause some aspects of the simulation to run more efficiently. Note that $|\mathcal{R}| = |V|$ if $\mathcal{G}$ is simply the dual graph of $\mathcal{R}$.

**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 $3p$. A disk has a complexity of 4 because it can be defined by a center point in $\mathbb{R}^3$ and a radius. Naturally, other choices are possible as well. Some navigation meshes have extra annotations, such as the maximum allowed radius of a character for an edge traversal [Kallmann 2014]. We will not include such annotations in this metric.

We measure three variants: 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. If a mesh has a small number of regions, a low region complexity, *and* high coverage, then it is a very efficient description of $\mathcal{E}_{free}$.

## 5.4 Performance

The final set of metrics concerns the practical performance of the navigation mesh implementations. One issue to take into account is that voxel-based methods perform more steps than exact methods. 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, 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 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 7 PC with a 3.20 GHz Intel i7-3930K CPU, an NVIDIA GeForce GTX 680 GPU, and 16 GB of RAM.

### 6.1 Navigation Meshes

We compare five state-of-the-art navigation mesh implementations and one extra baseline method. The first two navigation meshes are exact; the others are voxel-based and cover the full 3D pipeline. We currently include only the navigation meshes that are designed specifically for the environments described in Section 3, and for which we could obtain robust source code from their respective authors. Naturally, we encourage others to join this comparison.

Table 1 summarizes each navigation mesh based on the properties from Section 4. Most of these navigation meshes depend on various parameters. The parameter settings that we use in our experiments are listed in Appendix A in the supplementary file. Examples of the output for each method are shown in Figure 1.

**Local Clearance Triangulation (LCT).** The LCT [Kallmann 2014] subdivides a 2D environment of complexity $n$ into $\mathcal{O}(n)$ triangles by first computing a constrained Delaunay triangulation and then adapting it in $\mathcal{O}(n^2)$ worst-case time. The tested implementation runs in $\mathcal{O}(n\sqrt{n})$ expected time by using a special point-location method. These triangles are the regions of $\mathcal{R}$, and $\mathcal{G}$ is their dual graph. Triangle edges are annotated with clearance values to allow path planning for characters of an arbitrary radius. The LCT also supports dynamic updates. An extension to MLEs has not been developed, but an approach similar to the Explicit Corridor Map (described next) should be possible.

The LCT uses line segments as input and output, so our benchmark program needs to translate between walkable regions and boundary representations. These steps will not be included in our time measurements.

**Explicit Corridor Map (ECM).** The ECM [van Toll et al. 2011] 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, and each edge $E$ is a sequence of medial axis arcs. Certain medial axis points are annotated with their two nearest obstacle points, which induces a subdivision of $\mathcal{E}_{free}$ into polygonal regions.

The ECM enables path planning for characters of any radius, and the nearest obstacle in any region can be found in constant time. 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, it has size $\mathcal{O}(kn)$ and can be computed in $\mathcal{O}(kn \log n)$ time by iteratively opening the connections. In exchange for its advantages, the ECM is mathematically more complex than e.g. a triangulation.

**Clearance Disk Graph (CDG).** Pettré et al. [2005] presented the first *voxel-based* navigation mesh for 3D environments. In this paper, we refer to it as the Clearance Disk Graph (CDG). The CDG uses voxelization to approximate the areas where characters can stand. Next, the voxels are extracted for which the clearance is locally largest. These form an approximation of the medial axis of $\mathcal{E}_{free}$, and each cell represents an obstacle-free disk. A subset of these disks is chosen as the set of regions $\mathcal{R}$, and the graph $\mathcal{G}$ describes the disks and their overlap. 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 $S$.

**Recast.** The Recast Navigation toolkit [Mononen 2014] is a popular choice for game development that is also used in the Unity3D game engine [2016]. Like the CDG, it uses voxelization to approximate $\mathcal{E}_{free}$. However, Recast converts the walkable voxels to non-overlapping convex polygonal regions. This conversion involves many parameters that the user needs to tweak to get the best results. One 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 used for path planning, and a detailed mesh with more accurate height differences. We will use the coarse mesh to determine $\mathcal{R}$ and $\mathcal{G}$, and the detailed mesh to measure coverage.

**NEOGEN.** The NEOGEN method [Oliva and Pelechano 2013b] also starts with voxelization, but it groups walkable voxels into 2D layers. Next, the method obtains a more precise floorplan for each layer 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 [Oliva and Pelechano 2011] is used to compute the final navigation mesh. This 2D algorithm subdivides the layer into convex polygons in $\mathcal{O}(nr)$ time, where $r < n$ is the number of convex polygon vertices in the input. In our experiments, for simplicity, we will use the 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. Clearance information can also be added to the navigation mesh if desired [Oliva and Pelechano 2013a].

**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 in 2D environments and WEs [Sturtevant 2011], and they are a common choice for games that are grid-based by design [Sturtevant 2012]. Another advantage is that algorithms such as A* search can be optimized for grids [García et al. 2014; Harabor and Grastien 2011; Lee and Lawrence 2013; Sturtevant and Rabin 2016]. Many variants of A* are either designed with grids in mind [Botea et al. 2004] or explained in terms of grids [Koenig and Likhachev 2002; Koenig et al. 2004; Likhachev et al. 2005]. However, grids are typically more dense than other navigation meshes, which makes them less appropriate for planning many paths in real-time.

### 6.2 Implementation

We have converted each navigation mesh program to a stand-alone 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. Also, the CDG requires the walkable surfaces to be visible from all sides; to ensure this, we extrude all input polygons downwards by a small amount.

An important 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

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

**Table 1:** *Overview of the navigation meshes compared in this paper. For the rightmost columns, '**+**' means that a property is supported in the current implementation, '**+/-**' means that a property could be added in theory, and '**-**' means that a property is not supported by definition.*

erroneous points of $\mathcal{R}$ from getting mapped onto surfaces that are too far away. We choose a value of $T = 1$m because the vertical clearance is at least 2 m 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.)

We have implemented our coverage metrics using a CGAL-based program [CGAL 2016] that can compute the intersection of two OBJ files based on the threshold distance $T$. For this program, 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.

## 6.3 Environments

We have computed navigation meshes for the 2D and 3D input environments shown in Figures 3 and 4, which range in scale and complexity. Due to space constraints, we focus on environments that have appeared in previous publications. To test for scalability, we have also added randomly generated 2D mazes of various sizes, inspired by Sturtevant [2012]. In an extended publication, we will propose a more comprehensive set of benchmark environments. We have converted each environment to a clean representation of $\mathcal{E}_{free}$, subdivided into layers whenever necessary. The corresponding OBJ files are included in the supplementary file of this paper.

## 6.4 Results

The full set of results can be found in the attached supplementary file. Tables 2 and 3 show the results for coverage and connectivity, and Tables 4 and 5 show the results for complexity and performance. We will now highlight a number of observations.

**Coverage.** We have chosen our parameters to maximize coverage (see Appendix A). Still, in terms of absolute values, the voxel-based methods sometimes missed large areas or covered large incorrect parts, up to hundreds of square meters in large enviroments. However, the *relative* coverage was still high (typically over 90%).

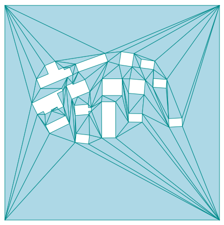
The maze environments are an exception: even though their free space was perfectly aligned with grid cells of $1 \times 1$m, the CDG and Recast could not capture them accurately. However, these mazes are quite detailed relative to their size, so a finer grid resolution could improve the results. NEOGEN generally yielded better coverage due to its extra processing step per layer. 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.

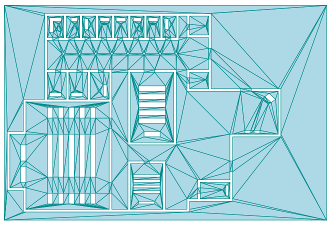
**Connectivity.** Recast and NEOGEN captured connectivity quite well for most environments, except in the mazes where each accidental gap causes parts to become disconnected. For the CDG, the graph usually contained many connected components, and gaps in the covered space led to a large number of boundaries. The grid also contained unexpected gaps at times, due to small errors in the current implementation. Still, the grid method works sufficiently well for the purpose of a comparison.

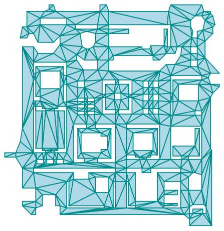
It is motivating to see that bad connectivity values corresponded to navigation meshes that were also *visually* incorrect. For example, Recast gave overlapping regions in some mazes, and it accidentally connected layers vertically in the *Tower* environment. Still, we acknowledge that metrics can never fully replace visual inspection.

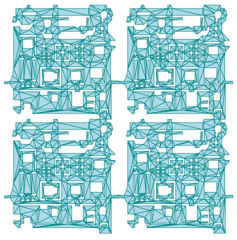
**Complexity.** NEOGEN typically yielded the smallest graph in exchange for the highest average region complexity. This makes sense because the algorithm is deliberately designed to produce a small number of regions [Oliva and Pelechano 2011]. Its convexity relaxation parameter could enlarge this effect. The ECM often produced smaller graphs than the LCT, while the LCT usually had a lower total region complexity.

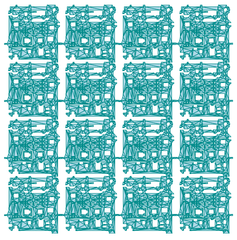
Recast appears to average between graph complexity and region complexity using our current settings. The method contains several parameters (such as the maximum number of vertices per region) with which this balance can be controlled. Recast and NEOGEN may not always capture all details of $\mathcal{E}_{free}$, but this does generally lead to simpler navigation meshes. This is useful for applications in which low storage requirements and fast path planning are more important than perfect coverage. In fact, exact methods could also benefit from a pre-processing step that simplifies $\mathcal{E}_{free}$.

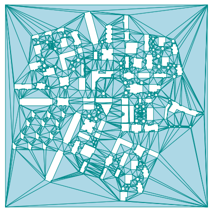

**(a)** *Military* ($n = 86$, $d = 200 \times 200$)

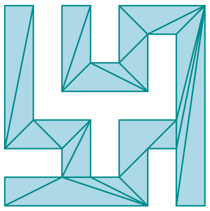

**(b)** *University* ($n = 560$, $d = 120 \times 80$)

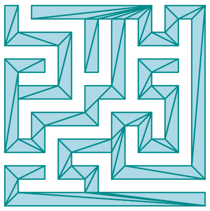

**(c)** *Zelda* ($n = 437$, $d = 100 \times 100$)

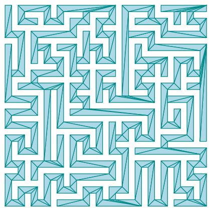

**(d)** *Zelda2x2* ($n = 1{,}768$, $d = 200 \times 200$)

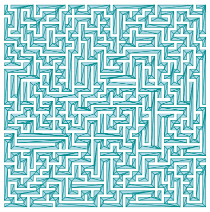

**(e)** *Zelda4x4* ($n = 7{,}072$, $d = 400 \times 400$)


**(f)** *City* ($n = 2{,}094$, $d = 500 \times 500$)


**(g)** *Maze8* ($n = 28$, $d = 7 \times 7$)


**(h)** *Maze16* ($n = 82$, $d = 15 \times 15$)


**(i)** *Maze32* ($n = 356$, $d = 31 \times 31$)


**(j)** *Maze64* ($n = 1{,}390$, $d = 63 \times 63$)

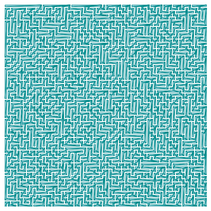
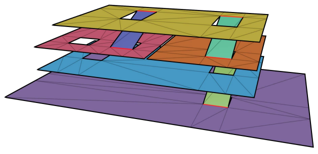
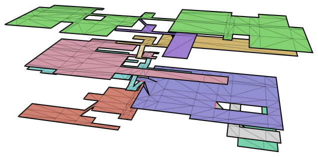
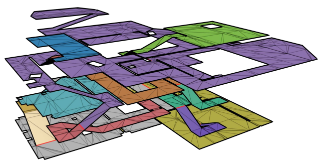
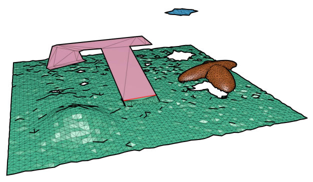

**(k)** *Maze128* ($n = 5{,}574$, $d = 127 \times 127$)

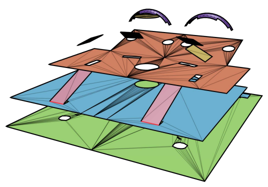
**Figure 3:** *Top views of the 2D environments used in our experiments. The number of polygon vertices* $n$ *and the physical dimensions* $d$ *(in meters) are shown in brackets.*

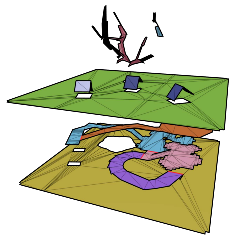

**(a)** *ParkingLot* ($n = 120$, $d = 36 \times 21 \times 9$)

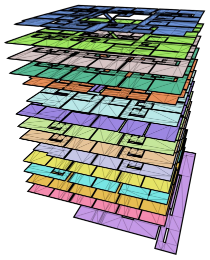

**(b)** *Library* ($n = 489$, $d = 59.5 \times 23.9 \times 21.1$)


**(c)** *Oilrig* ($n = 1{,}644$, $d = 273.6 \times 225.6 \times 84.4$)


**(d)** *Neogen1* ($n = 5{,}473$, $d = 63.5 \times 63.5 \times 15.7$)


**(e)** *Neogen2* ($n = 1{,}089$, $d = 63.5 \times 63.5 \times 31.1$)


**(f)** *Neogen3* ($n = 672$, $d = 63.5 \times 63.5 \times 57.4$)


**(g)** *Tower* ($n = 9{,}677$, $d = 34.8 \times 37.3 \times 39.2$)

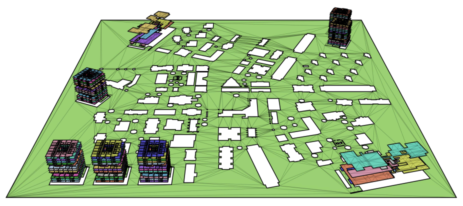

**(h)** *BigCity* ($n = 61{,}785$, $d = 500 \times 500 \times 39.2$)

**Figure 4:** *Renders 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.*

As expected, our grid implementation always 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.

**Performance.** The LCT implementation was by far the fastest in all environments, although it required pre-processing that we have not included in our measurements. As expected, exact methods scaled better to large environments than voxel-based methods: while the LCT and ECM remained fast, the running times increased strongly for Recast and the CDG in particular. NEOGEN was usually the fastest voxel-based method. The *BigCity* environment challenged the limits of all voxel-based methods: only Recast could produce a navigation mesh using our settings. Increasing the voxel resolution 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.

The 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, this indicates that obtaining $\mathcal{E}_{free}$ *without* voxels is an interesting topic for future work.

## 7 Discussion

A limitation of our comparison lies in the current set of input environments. We have focused on examples from previous publications; these are all realistic scenarios that have been considered interesting before. 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 for the same input. The goal of this paper was not to provide an exhaustive set of environments, or to expose all strengths and weaknesses of an implementation. Ultimately, it would be good to create an open database of input scenarios for researchers to use, similarly to the ones that currently exist for local character steering [Singh et al. 2009] and grid-based A* search [Sturtevant 2012].

We would also like to investigate more types of metrics. 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 $\mathcal{R}$? Another option is to look at the *quality* of these routes: how short are they, and how well do they correspond to real-life behavior? If shortness is important, a dense grid may yield better results than a navigation mesh with a small dual graph. Ultimately, for real-world applications, we would like to quantify how well a navigation mesh captures the navigation abilities of real humans. This is a challenging direction for future work. We expect that not everything can be analyzed mechanically, and that user studies will also be required.

Finally, to simplify the comparison, we have chosen a single set of parameter settings for all methods. It would be interesting to see how different settings influence the results for each method, and how these settings can be optimized for particular metrics. For example, Oliva and Pelechano have discussed how the voxel size affects the results of Recast and NEOGEN [2013b]. We would like to combine such ideas with the quantitative metrics of our paper.

## 8 Conclusions and Future Work

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

While we intend to use more environments, metrics, and settings, our results already suggest interesting properties. Voxel-based methods can be tuned to yield good coverage, but they do not always preserve connectivity, and their construction time 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.

The goal of this paper was not to find 'the best' navigation mesh, but to develop a way of comparing navigation meshes based on theoretical properties and quantitative metrics. Users may decide which properties and metrics are the most relevant for their application. We expect that this study will set a new standard for the evaluation and development of navigation meshes, and that it can help users choose an appropriate navigation mesh for particular applications.

Aside from the discussion points mentioned in Section 7, a topic for future work lies in developing *exact* algorithms that automatically extract the walkable space from arbitrary 3D input in real-time. Our experiments suggest that voxel-based approaches do not always preserve coverage and connectivity, and that they are not very scalable to large environments. However, an advantage of voxelization is that the input is automatically simplified to a certain level of precision. *Exact* filtering algorithms should yield a perfect representation of $\mathcal{E}_{free}$ within provable time bounds, but they may be sensitive to small details or imprecisions in the input (such as gaps or overlap). In the end, it may turn out that the best solution is to combine various approaches, e.g. a filtering algorithm without voxels that is based on rounded coordinates.

## Acknowledgements

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 and computing coverage using their software.

## References

ALI, S., NISHINO, K., MANOCHA, D., AND SHAH, M. 2013. *Modeling, Simulation and Visual Analysis of Crowds: A Multidisciplinary Perspective*. Springer.

BERSETH, G., KAPADIA, M., AND FALOUTSOS, P. 2015. ACCLMesh: Curvature-based navigation mesh generation. In *Proc. 8th ACM SIGGRAPH Conf. on Motion in Games*, 97–102.

BOOST, 2015. The Boost C++ library. http://www.boost.org/.

BOTEA, A., MÜLLER, M., AND SCHAEFFER, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development 1*, 7–28.

BUDDE, S. 2013. *Automatic generation of jump links in arbitrary 3D environments for navigation meshes*. Master's thesis, Humboldt-Universität zu Berlin.

CGAL, 2016. The Computational Geometry Algorithms Library. http://www.cgal.org/.

CURTIS, S., BEST, A., AND MANOCHA, D. 2014. Menge: A modular framework for simulating crowd movement. Tech. rep., University of North Carolina at Chapel Hill.

DEUSDADO, L., FERNANDES, A. R., AND BELO, O. 2008. Path planning for complex 3D multilevel environments. *Proc. 24th Spring Conf. on Computer Graphics*, 187–194.

GARCÍA, F. M., KAPADIA, M., AND BADLER, N. M. 2014. GPU-based dynamic search on adaptive resolution grids. In *Proc. IEEE Int. Conf. on Robotics and Automation*, 1631–1638.

GERAERTS, R. 2010. Planning short paths with clearance using Explicit Corridors. In *Proc. IEEE Int. Conf. on Robotics and Automation*, 1997–2004.

HALE, D. H., AND YOUNGBLOOD, G. M. 2009. Full 3D spacial decomposition for the generation of navigation meshes. In *Proc. 5th Artificial Intelligence and Interactive Digital Entertainment Conf.*, 143–147.

HALE, D. H., YOUNGBLOOD, G. M., AND DIXIT, P. N. 2008. Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds. In *Proc. 4th Artificial Intelligence and Interactive Digital Entertainment Conf.*, 173–178.

HARABOR, D., AND GRASTIEN, A. 2011. Online graph pruning for pathfinding on grid maps. In *Proc. 52th AAAI Conf. on Artificial Intelligence*, 1114–1119.

HART, P., NILSSON, N., AND RAPHAEL, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 2, 100–107.

HILLEBRAND, A. 2012. *Separating a polygonal environment into a multi-layered environment*. Master's thesis, Utrecht University, The Netherlands.

KALLMANN, M. 2014. Dynamic and robust Local Clearance Triangulations. *ACM Transactions on Graphics 33*, 5.

KAPADIA, M., PELECHANO, N., ALLBECK, J., AND BADLER, N. I. 2015. *Virtual Crowds: Steps Toward Behavioral Realism*. Morgan & Claypool Publishers.

KAVRAKI, L. E., ŠVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12*, 4, 566–580.

KOENIG, S., AND LIKHACHEV, M. 2002. D* Lite. In *Proc. AAAI Conf. of Artificial Intelligence*, 476–483.

KOENIG, S., LIKHACHEV, M., AND FURCY, D. 2004. Lifelong Planning A*. *Artificial Intelligence 155*, 1-2, 93–146.

LAMARCHE, F. 2009. TopoPlan: a topological path planner for real time human navigation under floor and ceiling constraints. *Computer Graphics Forum 28*, 2, 649–658.

LAVALLE, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

LEE, W., AND LAWRENCE, R. 2013. Fast grid-based path finding for video games. In *Advances in Artificial Intelligence*, vol. 7884 of *Lecture Notes in Computer Science*. Springer, 100–111.

LIKHACHEV, M., FERGUSON, D., GORDON, G., STENTZ, A., AND THRUN, S. 2005. Anytime Dynamic A*: An anytime, replanning algorithm. In *Proc. Int. Conf. on Automated Planning and Scheduling*, 262–271.

LOPEZ, T., LAMARCHE, F., AND LI, T.-Y. 2012. Space-time planning in changing environments: using dynamic objects for accessibility. *Computer Animation and Virtual Worlds 23*, 87–99.

LOZANO-PEREZ, T. 1983. Spatial planning: A configuration space approach. *IEEE Transactions on Computing 32*, 2, 108–120.

MONONEN, M., 2014. Recast Navigation. https://github.com/memononen/recastnavigation/.

OLIVA, R., AND PELECHANO, N. 2011. Automatic generation of suboptimal navmeshes. In *Proc. 4th Int. Conf. on Motion in Games*, 328–339.

OLIVA, R., AND PELECHANO, N. 2013. A generalized exact arbitrary clearance technique for navigation meshes. In *Proc. 6th Int. Conf. on Motion in Games*, 103–110.

OLIVA, R., AND PELECHANO, N. 2013. NEOGEN: Near optimal generator of navigation meshes for 3D multi-layered environments. *Computers & Graphics 37*, 5, 403–412.

PELECHANO, N., ALLBECK, J. M., KAPADIA, M., AND BADLER, N. I. 2016. *Simulating Heterogeneous Crowds with Interactive Behaviors*. CRC Press.

PETTRÉ, J., LAUMOND, J., AND THALMANN, D. 2005. A navigation graph for real-time crowd animation on multilayered and uneven terrain. In *Proc. 1st Int. Workshop on Crowd Simulation*, 81–89.

POLAK, R. M. 2016. *Extracting walkable areas from 3D environments*. Master's thesis, Utrecht University.

RICKS, B. C., AND EGBERT, P. K. 2014. A whole surface approach to crowd simulation on arbitrary topologies. *IEEE Trans. Visualization and Computer Graphics 20*, 159–171.

SINGH, S., KAPADIA, M., FALOUTSOS, P., AND REINMAN, G. 2009. An open framework for developing, evaluating, and sharing steering algorithms. In *Proc. 2nd Int. Workshop on Motion in Games*, 158–169.

SNOOK, G. 2000. Simplified 3D movement and pathfinding using navigation meshes. In *Game Programming Gems*, M. DeLoura, Ed. Charles River Media, 288–304.

STURTEVANT, N., AND RABIN, S. 2016. Canonical orderings on grids. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 683–689.

STURTEVANT, N. 2011. A sparse grid representation for dynamic three-dimensional worlds. In *Proc. AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment*.

STURTEVANT, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games 4*, 2, 144–148.

THALMANN, D., AND MUSSE, S. R. 2013. *Crowd Simulation*, 2 ed. Springer.

VAN TOLL, W. G., COOK IV, A. F., AND GERAERTS, R. 2011. Navigation meshes for realistic multi-layered environments. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 3526–3532.

VAN TOLL, W., JAKLIN, N., AND GERAERTS, R. 2015. Towards believable crowds: A generic multi-level framework for agent navigation. In *ASCI.OPEN*.

TOZOUR, P. 2002. Building a near-optimal navigation mesh. In *AI Game Programming Wisdom*, S. Rabin, Ed. Charles River Media, 171–185.

UNITY3D GAME ENGINE, 2016. http://www.unity3d.com/.