Annotating traversable gaps in walkable environments

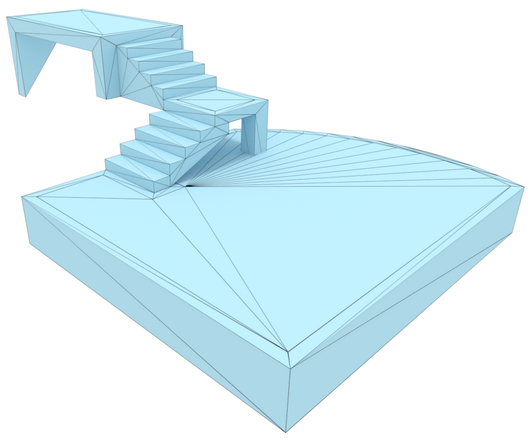
Utrecht University, The Netherlands

Jordi L. Vermeulen, Arne Hillebrand and Roland Geraerts

Utrecht University

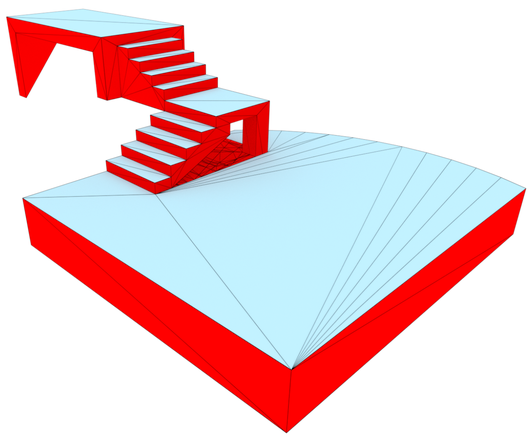
To allow efficient path planning in 3D environments, a navigation mesh is typically generated. To generate such a mesh, we need to find the parts of the environment that an agent can walk on, called the **walkable environment**. However, such an environment might contain gaps that an agent could easily step across. We present an exact, heuristic algorithm for detecting and filling such gaps, and compare our results to Recast, a voxel-based method for navigation mesh generation.

Walkable environments



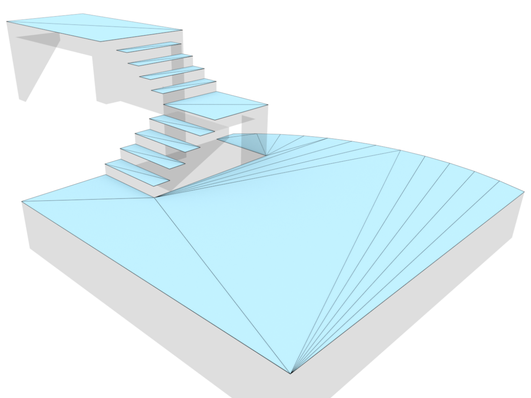
Model acquisition

We first obtain a model of the environment we want our agents to navigate. Some methods of model acquisition may leave gaps in the model, for instance when the model is reconstructed from a point cloud, or converted from smooth patches.



Filtering

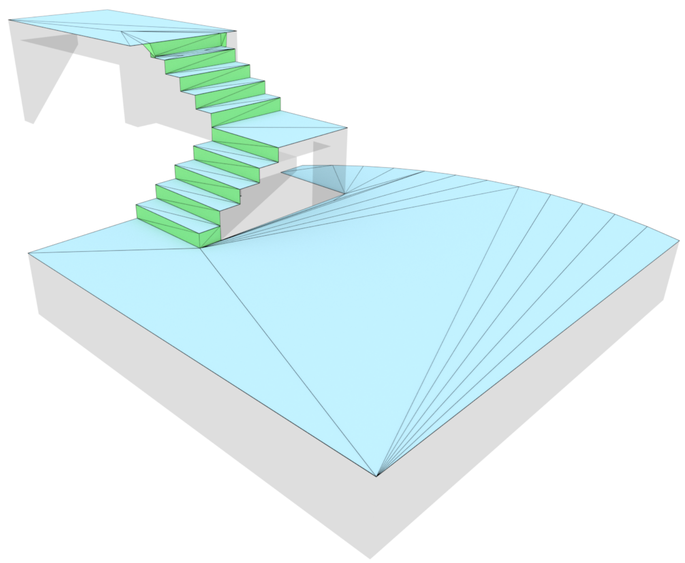
To find the walkable environment, we filter out the parts that are too steep or do not have enough vertical clearance for our agents to stand on.



Walkable environment

The resulting walkable environment may contain gaps due to vertical steps present in the environment, or because the surface itself contains holes (e.g. a metal grate).

Traversable gaps



Our algorithm

Our algorithm follows four basic steps:

1) Find cycles of boundary edges.
2) Detect gaps between boundary edges.
3) Connect boundary edges in different cycles.
4) Connect boundary edges in the same cycle.

Detection of gaps is done between all boundary edges within a given distance d_{max} of each other.

We prioritise connections between different components of the walkable environment, so we use the detected gaps to first connect different cycles. This is done heuristically based on which edge is closest when projected onto the ground plane.

After connecting different cycles, we use the gaps detected within the same cycle to fill holes. For this, we use only those parts of the hole that were not already connected to a different cycle, to prevent the introduction of singular edges.

Experiments and results

Implementation

We implemented our algorithm in C++, making use of CGAL for exact geometric computations. We tested our algorithm on a variety of artificial and real-world scenarios. We also compared our algorithm with Recast, a popular voxel-based method for generating navigation meshes from 3D environments.

Comparison

As our implementation computes both the walkable environment and the gaps exactly, our results are generally cleaner and more precise than those obtained with Recast. However, our implementation is orders of magnitude slower and uses much more memory.

Limitations

Our method currently does not properly handle areas where more than two boundary cycles are within d_{max} of each other. This is because our heuristic only chooses a single gap on each part of a segment. This sometimes leads to residual holes in the result. This may also happen when two boundaries have very dissimilar shapes.

Future work

Our algorithm needs to robustly handle obstacles. We can also fix the problems with residual holes by looking at chains of boundary edges and applying techniques from mesh repair. Finally, our implementation should be optimised to allow the processing of larger environments.