Efficient Octree-based 3D Pathfinding

2024 IEEE Conference on Games (CoG)

Quentin Massonnat, Clark Verbrugge
McGill University Montreal, Canada

Introduction

- 3D virtual environments are commonplace in modern games. However movement planning is still largely 2D.
- Pathfinding in 3D is expensive. Full 3D pathfinding is heavy on memory and computation time, and real-time use is challenging.
- Exact shortest path in 3D is NP-hard.

Approach

- This work proposes a hybrid 3D pathfinding method by extending 2D graph-based navigation meshes to a hierarchical environment representation.
  - Hertel-Mehlhorn method to merge cells for a coarser map, reducing path cost.
  - A* search on transition graph between cells.
  - Visibility pruning and 3D funnel algorithm.

Building the navigation mesh

- This work uses an octree-based navigation mesh for 3D pathfinding.
- Starting from a root cube that recursively subdivides obstacle-intersecting cells until a defined minimize size is reached, we build an octree representation.
- To reduce the number of cells, we merge adjacent convex cells using a greedy method based on the Hertel-Mehlhorn approach.

Path Finding and Path Refinement

- This work performs 3D pathfinding by constructing a graph over valid, convex octree cells.
- Each node lies at the center of a transition surface between adjacent cells, and edges are weighted by Euclidean distance.
  - This work define as transition surfaces the surfaces connecting two adjacent valid cells.
- A* is used to find the shortest path on this graph.

Path Finding and Path Refinement

- To refine paths, this work implements two methods
  - Visibility-based pruning heuristic that skips intermediate nodes,
  - Extension of the 2D funnel algorithm to 3D using the octree as convex decomposition.

Algorithm 1 Pseudocode for the path pruning algorithm

Input: A path between two points s and t ($s = x_0, x_1, ..., x_{k-1}, x_k = t$)
Output: The pruned path

```
Let anchor = t and prunedPath = [t]
for i decreasing from k − 1 to 0 do
    if x_i is visible from anchor then
        Discard the node x_i
    else
        Add the node x_{i+1} to prunedPath and let it be the new anchor
    end if
end for
Add s to prunedPath
return prunedPath in reverse order
```

Path Finding in Dynamic 3D Environments

- In dynamic scenes we track moving obstacles and apply local octree updates.
  - If a valid cell(none obstacle) becomes obstructed, it is recursively split until the target granularity is reached.
  - Conversely, when a cell(Obstacles were) becomes free, we attempt to merge its siblings to restore the parent.
- After update, corresponding changes are applied to the movement graph.

Experiments and Results

- Datasets: 6 handmade scenes and 19 Warframe voxel maps.
  - Warframe voxel maps come with 10,000 “test scenarios”

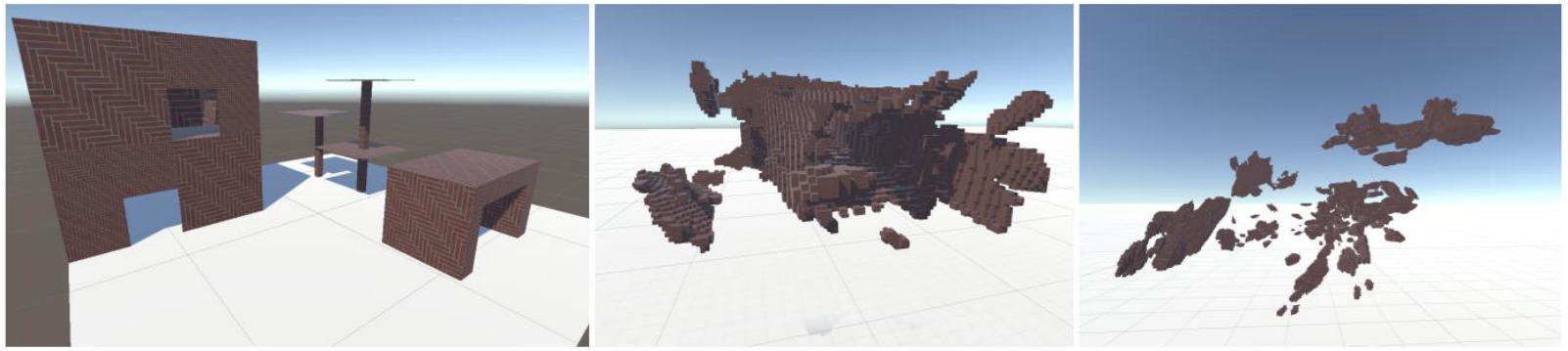


Fig. 1. Snapshot of the *Industrial* map from our handmade dataset (left), and the *Complex* (middle) and *Full4* (right) maps from the *Warframe* dataset.

Impact of Merging and Path Refinement

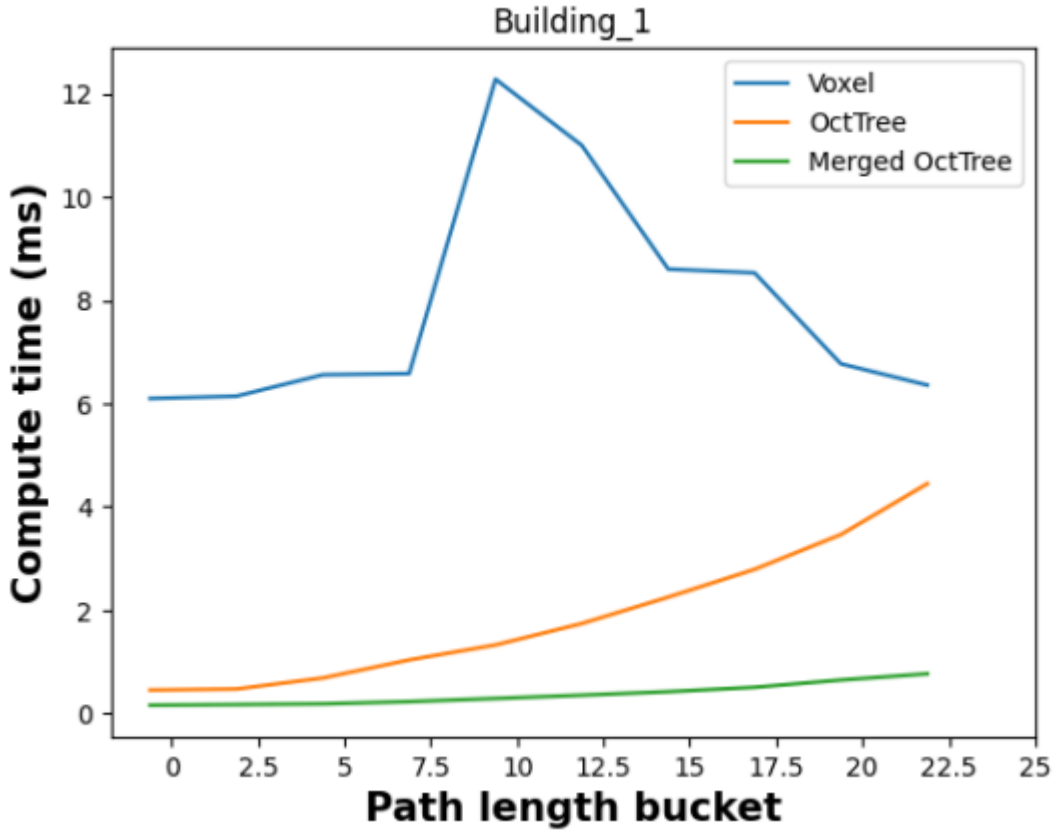
- Merging significantly reduces the number of valid octree cells

TABLE I
COMPARISON OF THE NUMBER OF VALID CELLS ACROSS SEVERAL MAPS USING EITHER THE VOXEL BASELINE, THE REGULAR, OR MERGED OCTREE.

Map name	Voxel	Unmerged	Merged
Building_1	28,190	4,226	**303**
Building_2	28,628	3,975	**147**
Building_3	29,816	4,243	**182**
Cave	29,510	8,190	**316**
Industrial	31,833	1,671	**157**
Zigzag	31,488	2,130	**52**
Complex (Warframe)	8.3M	41,385	**10,552**

Impact of Merging and Path Refinement

- The merged octree is faster than other approaches.



Experiments and Results

- This work evaluated this method by adding moving obstacles to existing datasets.

TABLE II

AVERAGE COST OF THE OCTREE UPDATE, GRAPH UPDATE, AND PATH RECOMPUTING IN MILLISECONDS OVER 1,000 UPDATES ON THE INDUSTRIAL MAP, USING EITHER REGULAR (O) OR MERGED OCTREES (MO), AND LOCAL GRAPH UPDATES (LGU) OR REBUILDING THE GRAPH

Update method	Octree update	Graph update	Path recomputing	Total time
O + no LGU	1.36	0.92	1.82	4.10
O + LGU	**0.22**	**0.03**	1.26	1.51
MO + no LGU	0.44	0.34	0.72	1.50
MO + LGU	0.39	**0.03**	**0.55**	**0.97**

ɔxel
ɛets

Comparison with JPS-3D

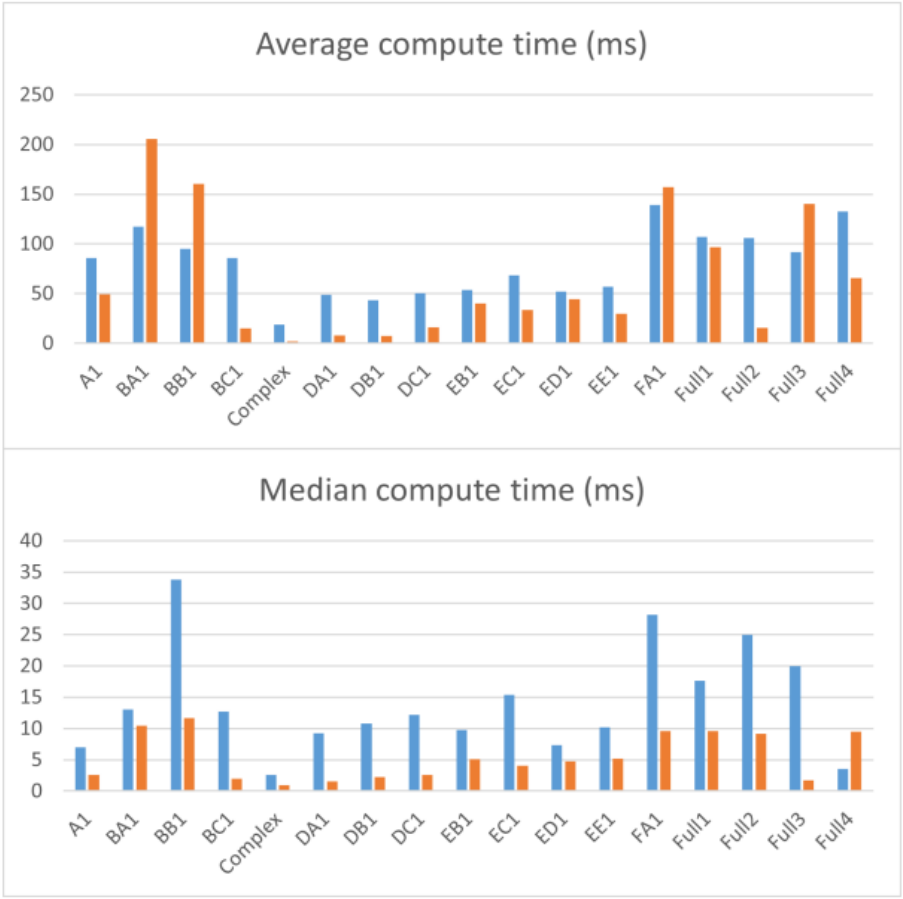
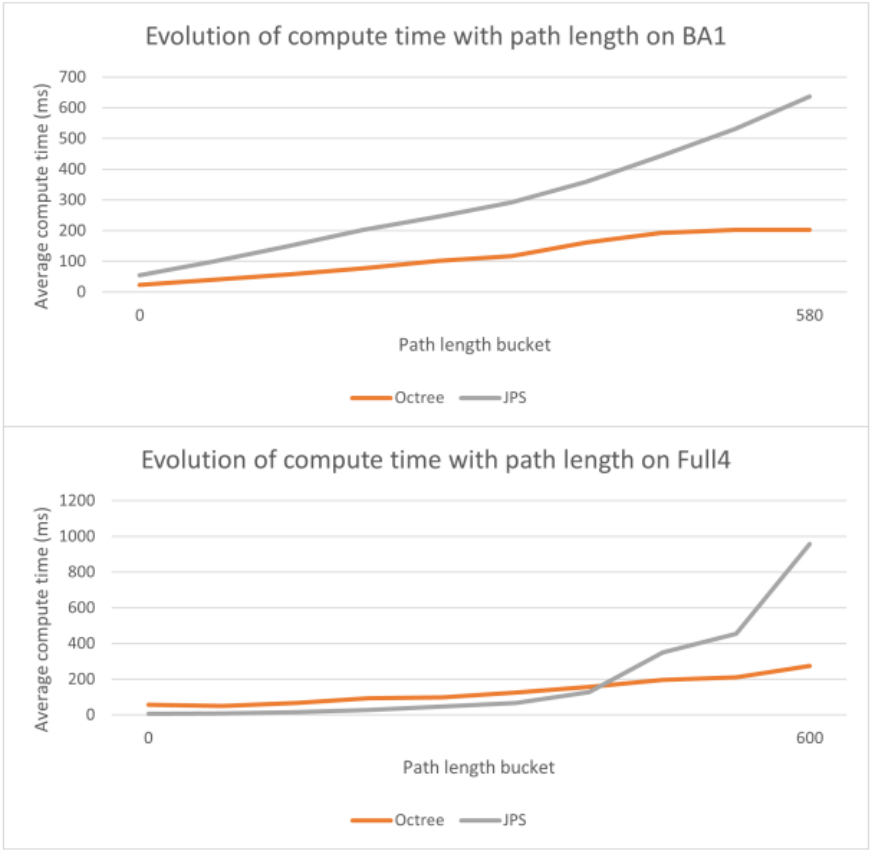


Fig. 5. Average and median compute time of merged octrees (in blue) and JPS-3D (in orange) on the 3D benchmark *Warframe*.

Comparison with JPS-3D



ıd Fig. 6. Evolution of the compute time with path length on the BA1 and Full4 maps.

Comparison with JPS-3D

- JPS is faster on average, but slower for longer paths.
- This work method finds shorter and more stable paths, especially in long or complex situations and it works in real time.

Conclusion

- Combining octrees with greedy merging enables real-time 3D pathfinding.
- These path refinement cuts path length by up to 10%
- This method is better than JPS-3D on long paths
- This work supports dynamic environments via local updates and repair-based merging
- Future work:
  - Flexible initial octree shape, for example rectangular base
  - Incorporate agent constraints (turning radius, smooth paths via Bezier)

Thank you for listening