Pathfinding in 3D Space

CHIA-MAN HUNG & RUOQI HE

Outline

- Introduction
- I. State of the art
- II. Algorithms
- III. Implementation in 3D space
- IV. Results
- Conclusion

Introduction

- Goal: Find the shortest paths efficiently in 3D space
- Applications: video games, drone navigation

I. State of the art

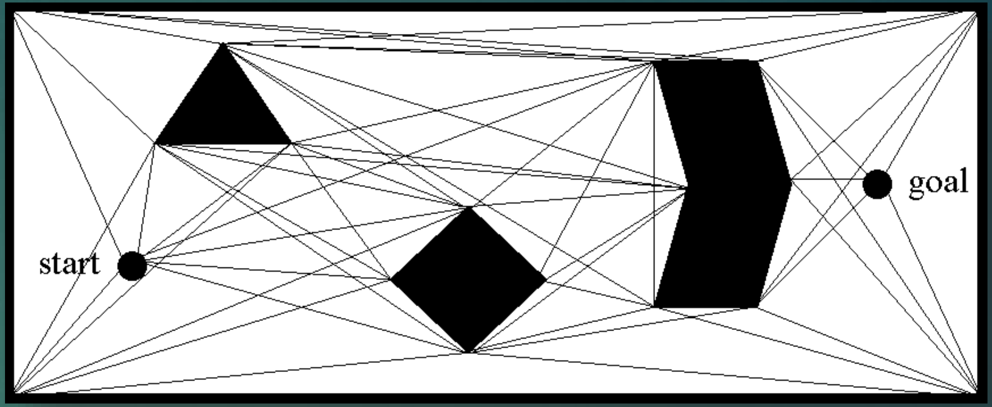
- Homeworld (1999) :

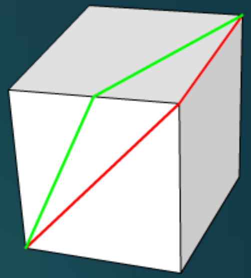
  First famous real-time strategy game with movement in 3D space

I. State of the art

- Shortest paths in a graph
  - Dijkstra (single source)
    - $O((|V|+|E|)\log(|V|))$
  - Bellman-Ford (single source, weighted directed graph)
    - $O(|V||E|)$
  - Floyd-Warshall (for all pairs of vertices, weighted graph , no negative cycle)
    - $O(|V|^3)$
  - A* (single source, single destination)
    - $O(n)$, n = length of the solution path => $O(|E|)$

I. State of the art

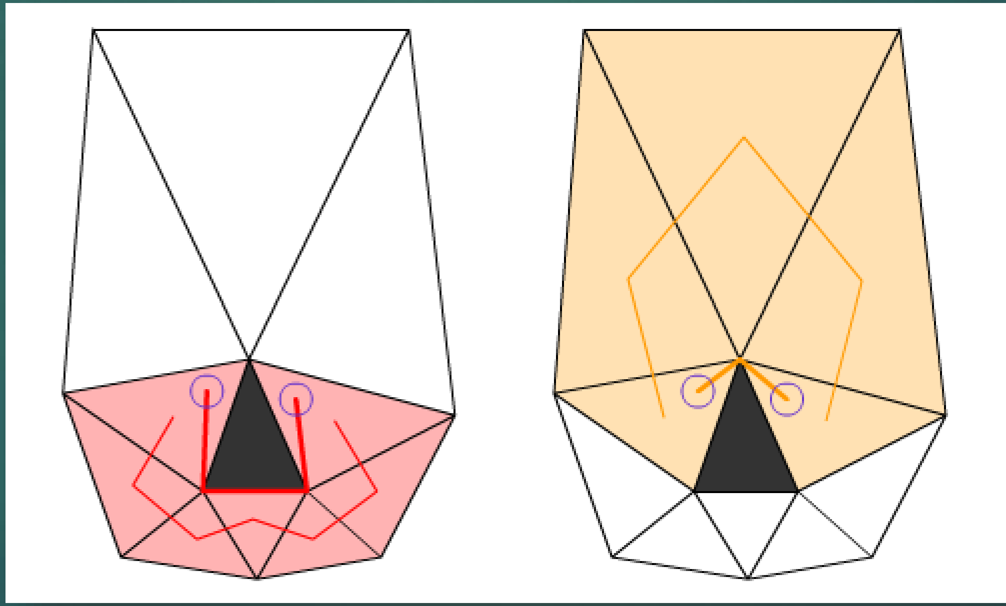




- 2D - exact
  - Visibility graph
  - Anya (2D grid)
- 2D - approximate
  - Waypoints
  - Navigation mesh + tunnel
  - Family of Theta*

Non-optimality

- Navigation mesh + tunnel



path found VS true shortest path

I. State of the art

- 3D surface - exact
  - Windows (Fast exact and approximate geodesics on meshes 2005 Surazhsky)
- 3D surface - approximate
  - Heat (Geodesics in heat 2013 Crane)
  - Fast-marching (1996 Sethian)

II. Algorithms

- World representation
  - Tetrahedralization
  - Convex decomposition
  - Grid
  - Octree

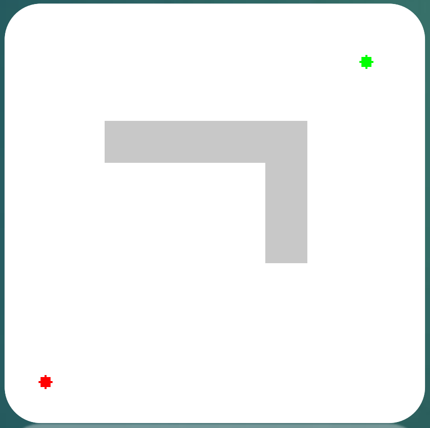
II. Algorithms

- A* (1968 Hart)

  h admissible if

  no over-estimation and

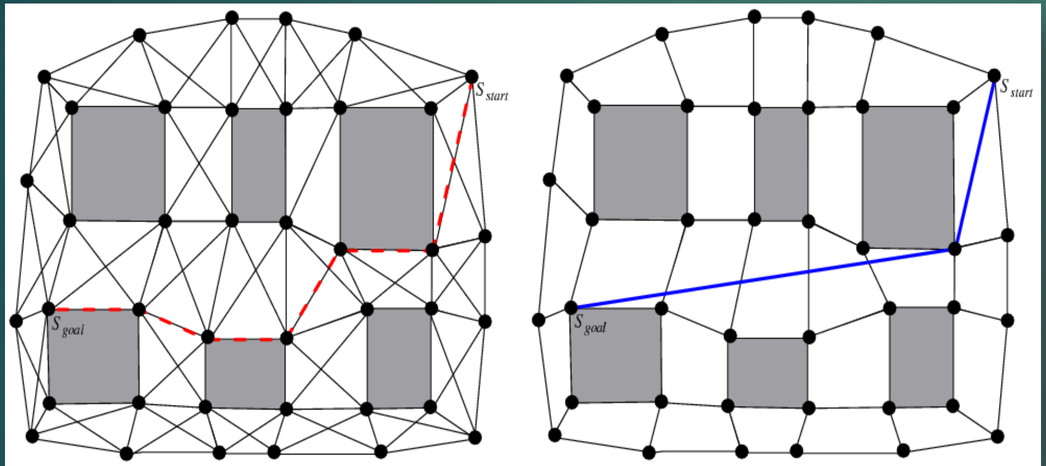
  h(y) <= h(x) + d(x, y)



```
Main()
    open := closed := ∅;
    g(s_start) := 0;
    parent(s_start) := s_start;
    open.Insert(s_start, g(s_start) + h(s_start));
    while open ≠ ∅ do
        s := open.Pop();
        if s = s_goal then
            return "path found";
        closed := closed ∪ {s};
        foreach s' ∈ nghbr_vis(s) do
            if s' ∉ closed then
                if s' ∉ open then
                    g(s') := ∞;
                    parent(s') := NULL;
                UpdateVertex(s, s');
    return "no path found";
end
UpdateVertex(s, s')
    g_old := g(s');
    ComputeCost(s, s');
    if g(s') < g_old then
        if s' ∈ open then
            open.Remove(s');
        open.Insert(s', g(s') + h(s'));
end
ComputeCost(s, s')
    /* Path 1 */
    if g(s) + c(s, s') < g(s') then
        parent(s') := s;
        g(s') := g(s) + c(s, s');
end
```

II. Algorithms

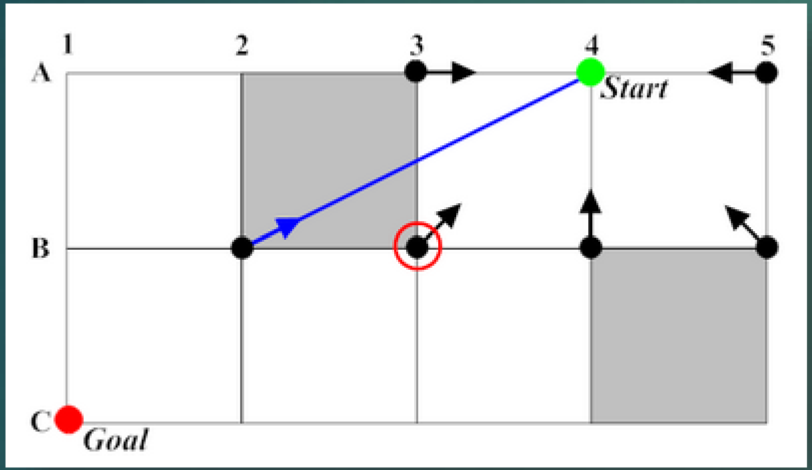
- Theta* (2007 Nash)




```
ComputeCost(s, s')
    if LineofSight(parent(s), s') then
        /* Path 2 */
        if g(parent(s)) + c(parent(s), s') < g(s') then
            parent(s') := parent(s);
            g(s') := g(parent(s)) + c(parent(s), s');
    else
        /* Path 1 */
        if g(s) + c(s, s') < g(s') then
            parent(s') := s;
            g(s') := g(s) + c(s, s');
end
```

II. Algorithms

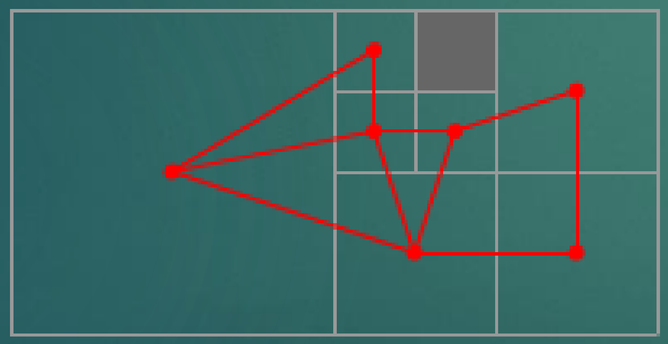
- Lazy Theta* (2010 Nash)



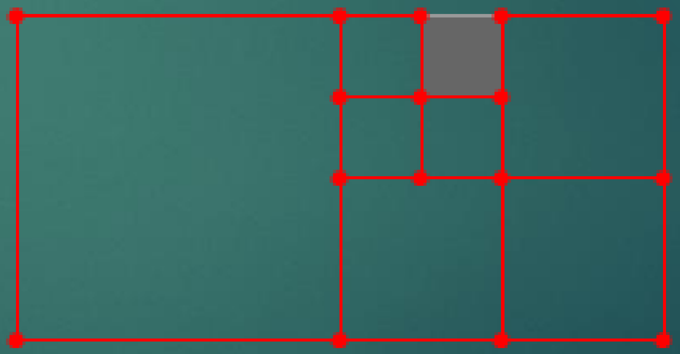

```
Main()
    open := closed := ∅;
    g(s_start) := 0;
    parent(s_start) := s_start;
    open.Insert(s_start, g(s_start) + h(s_start));
    while open ≠ ∅ do
        s := open.Pop();
        SetVertex(s);
        if s = s_goal then
            return "path found";
        closed := closed ∪ {s};
        foreach s' ∈ nghbr_vis(s) do
            if s' ∉ closed then
                if s' ∉ open then
                    g(s') := ∞;
                    parent(s') := NULL;
                UpdateVertex(s, s');
    return "no path found";
end
UpdateVertex(s, s')
    g_old := g(s');
    ComputeCost(s, s');
    if g(s') < g_old then
        if s' ∈ open then
            open.Remove(s');
        open.Insert(s', g(s') + h(s'));
end
ComputeCost(s, s')
    /* Path 2 */
    if g(parent(s)) + c(parent(s), s') < g(s') then
        parent(s') := parent(s);
        g(s') := g(parent(s)) + c(parent(s), s');
end
SetVertex(s)
    if NOT lineofsight(parent(s), s) then
        /* Path 1*/
        parent(s) :=
        argmin_{s'∈nghbr_vis(s)∩closed}(g(s') + c(s', s));
        g(s) :=
        min_{s'∈nghbr_vis(s)∩closed}(g(s') + c(s', s));
end
```

III. Implementation

- Octree construction
  - Triangle-cube intersection
  - Progressive octree
- Graph construction



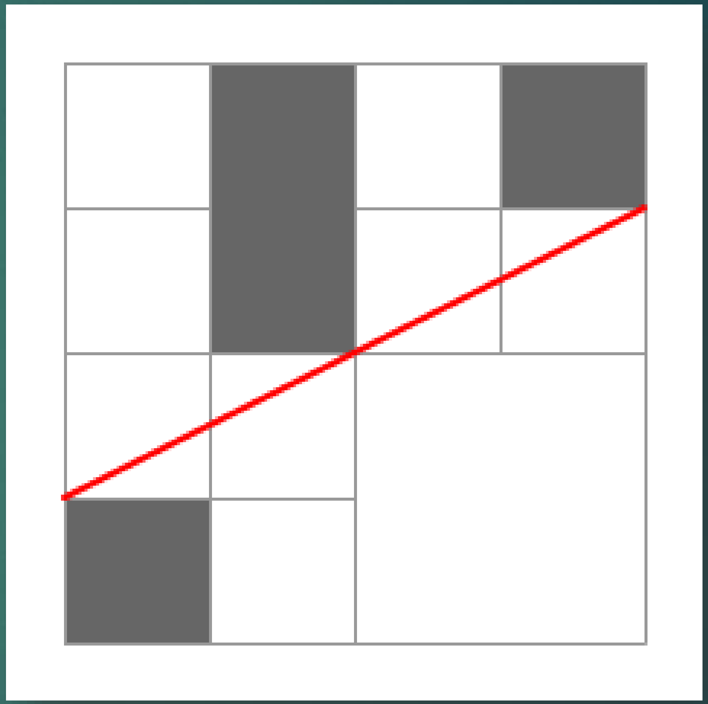
Dual graph (not standard)



Edge-corner

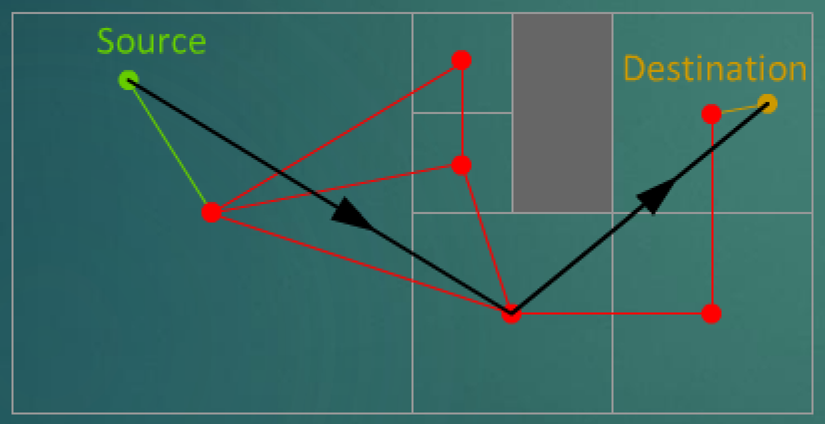
III. Implementation

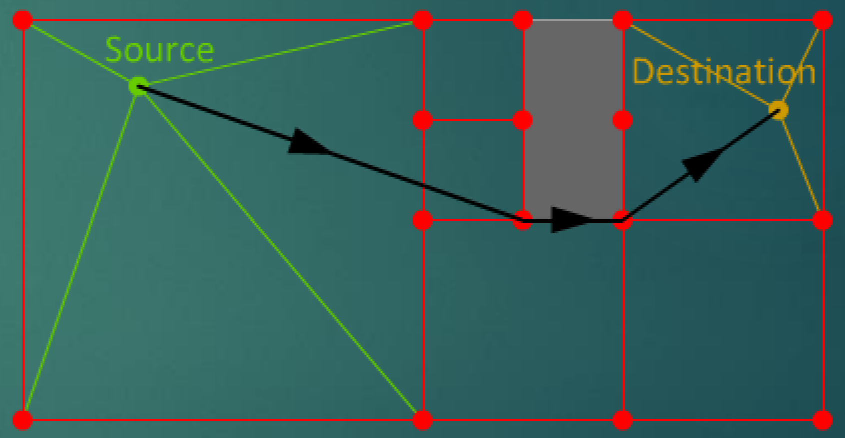
- Line of sight
  - Fast
  - Robust



III. Implementation

- Injection of source and destination





III. Implementation - Optimisation

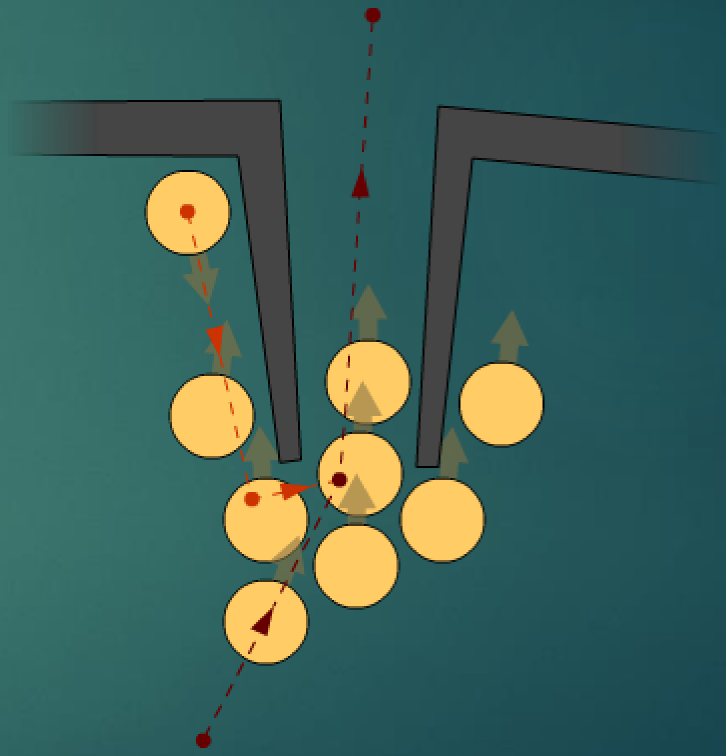
- Avoid exhaustive search
  - Precompute the connectedness of the graph nodes

III. Implementation - Optimisation

- Multisource
  - Reuse information

III. Implementation - Extension

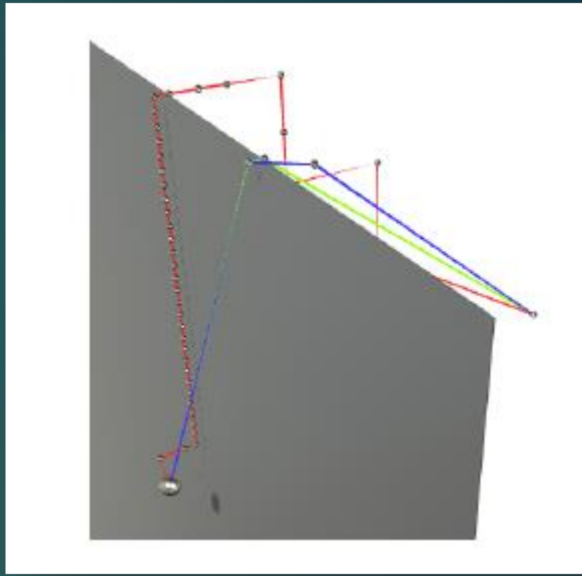
- Application in video games
  - Waypoints
  - Repulsive force
  - Replanification



IV. Results

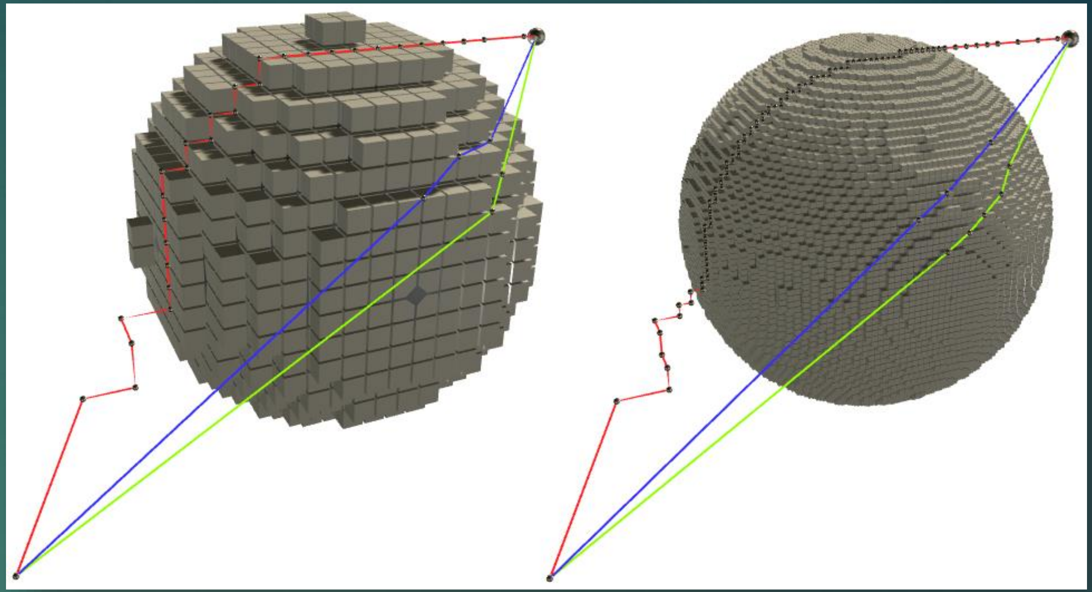
Data Structure	Algorithm	distance	time cost	distance	time cost
Octree	A*	125.18%	1.6ms	124.56%	4.5ms
	Theta*	101.58%	11.8ms	109.24%	53.0ms
	Lazy Theta*	101.73%	6.6ms	109.48%	22.1ms
Progressive Octree	A*	125.19%	4.6ms	126.27%	5.8ms
	Theta*	101.58%	21.5ms	101.63%	42.7ms
	Lazy Theta*	101.39%	10.6ms	102.27%	18.1ms

Table 1: Comparison - A single wall - Left: source sparse/ Right: destination sparse



Red: A*
Green: Theta*
Blue: Lazy Theta*

IV. Results



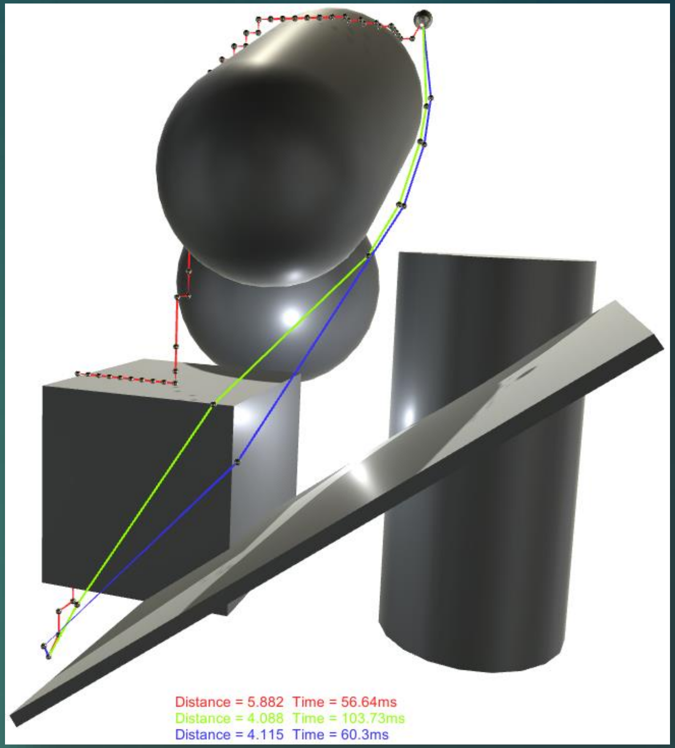
Red: A*
Green: Theta*
Blue: Lazy Theta*

Data Structure	Algorithm	distance	time cost	distance	time cost
Octree	A*	121.38%	1.5ms	121.03%	28.0ms
	Theta*	104.59%	6.7ms	102.71%	236.1ms
	Lazy Theta*	104.65%	4.2ms	102.34%	114.8ms
Progressive Octree	A*	127.43%	3.3ms	126.88%	55.6ms
	Theta*	103.49%	6.3ms	101.23%	229.4ms
	Lazy Theta*	103.68%	3.2ms	101.16%	108.8ms

Table 2: Comparison - A single sphere - Left: level = 7/ Right: level = 9

IV. Results

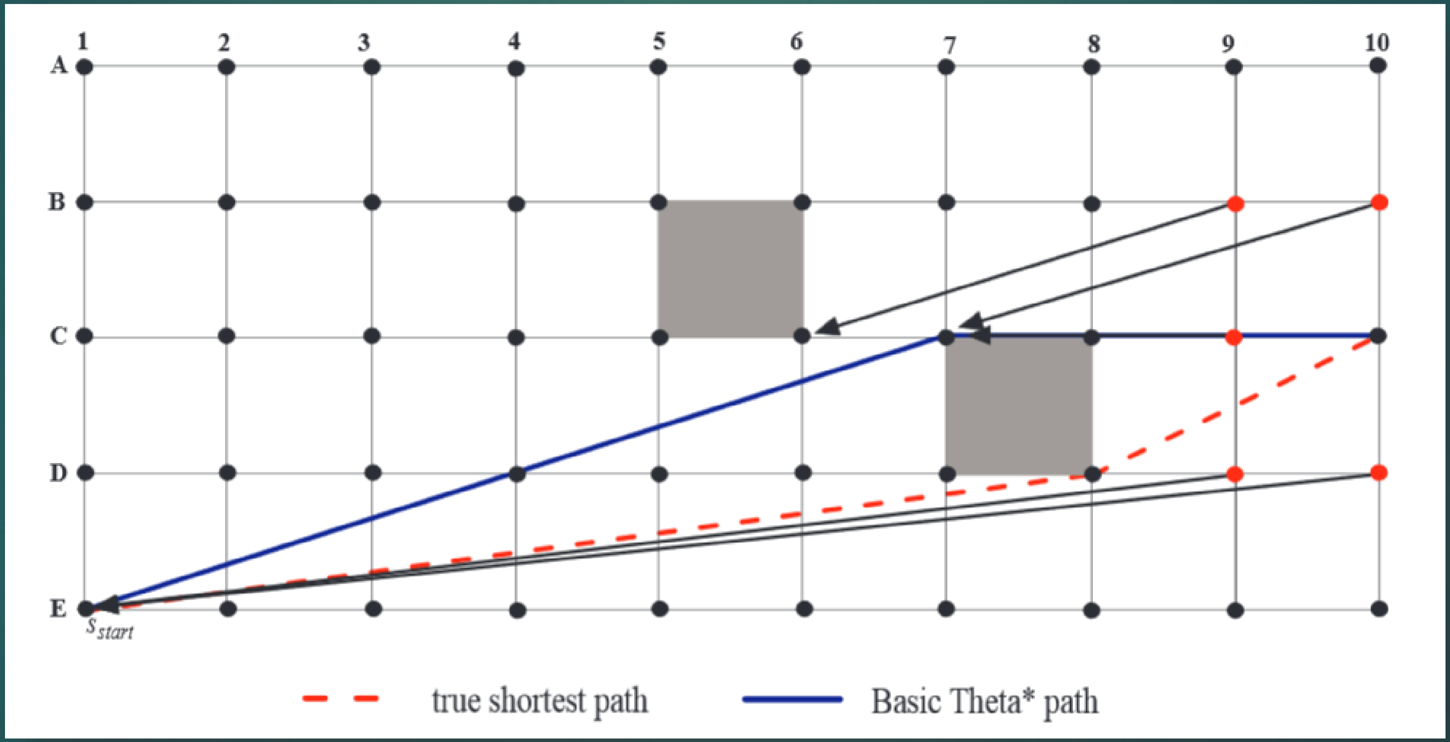
Red: A*
Green: Theta*
Blue: Lazy Theta*



Data Structure	Algorithm	distance	time cost	distance	time cost
Octree	A*	3.2472	9.5ms	3.3302	5.3ms
	Theta*	2.4108	23.9ms	2.4600	45.5ms
	Lazy Theta*	2.4135	9.5ms	2.4592	16.3ms
Progressive Octree	A*	3.3949	14.1ms	3.3222	7.15ms
	Theta*	2.4009	17.59ms	2.4158	43.1ms
	Lazy Theta*	2.4057	7.78ms	2.4205	14.6ms

Table 3: Comparison - A complex scene - Left: edge-corner graph/ Right: dual graph

Non-optimality of Theta*



Demo !

- Demo !
  - Demo !
    - Demo !
      - Demo !
        - Demo !
          - Demo !
            - Demo !
              - Demo !

Conclusion

- Exploration in a new domain
- Our proposition : Lazy Theta * + Progressive Octree + Edge-corner graph
- Possible Improvements
  - Distribution of computation at each frame
  - Other possibilities of h
  - Post-processing