Path Planning: Path Networks and NavMeshes

Grid Lattice Navigation – Pros

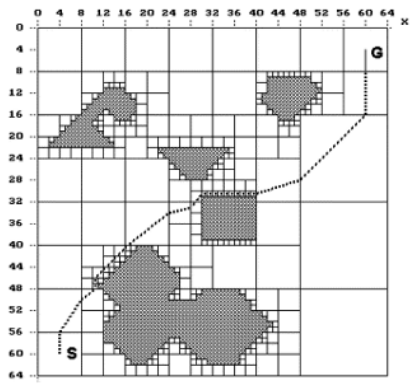
- Discrete space is uniform and simple
- Usually for games with discrete agent/NPC locations but can be used with continuous coordinates
- Easy to automatically generate
- Good for large number of units (esp. if units grid sized – easy to mark map cells as occupied or not)
- A* works well with grid

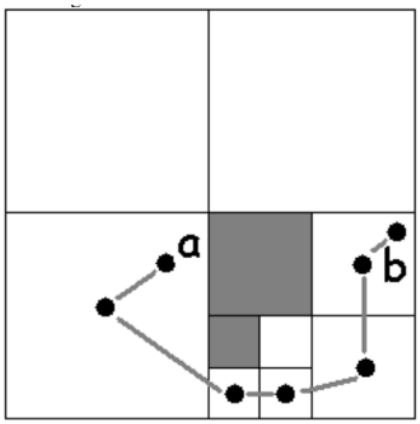
Grid Lattice Navigation – Cons

- Inefficient use of memory
- Resulting paths are jagged/blocky resulting in unnatural movement
- Doesn't work well to maximize navigable area (consider cells that are partially navigable)
- Search efficiency (must visit a lot of cells)
- Memory usage: 100x100 map == 10k nodes and ~78k edges

Quad Tree

- More efficient use of space than a grid





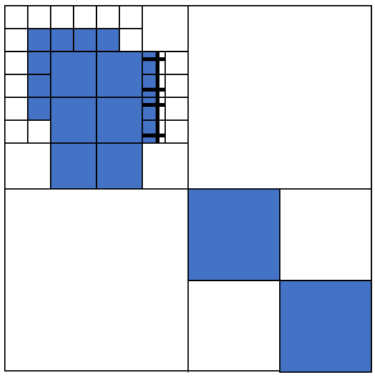
https://www.sciencedirect.com/science/article/pii/S0736584501000187

I. L. Davis, “Warp speed: Path planning for star trek: Armada,” presented at the AAAI Spring Symposium (AIIDE), 2000, pp. 18–21.

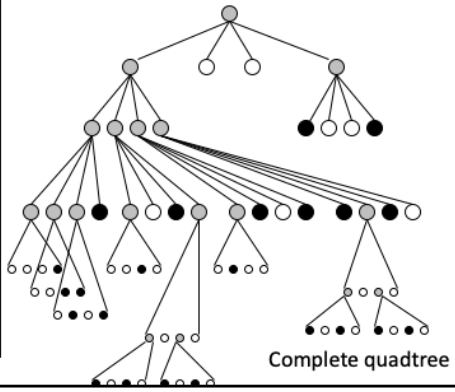
Quad Tree

- More complicated data structures and parsing
- Can result in undesirable quad tree subdivisions if obstacle details don't line up with quad tree cell boundaries

Space Representation



Equivalent quadtree



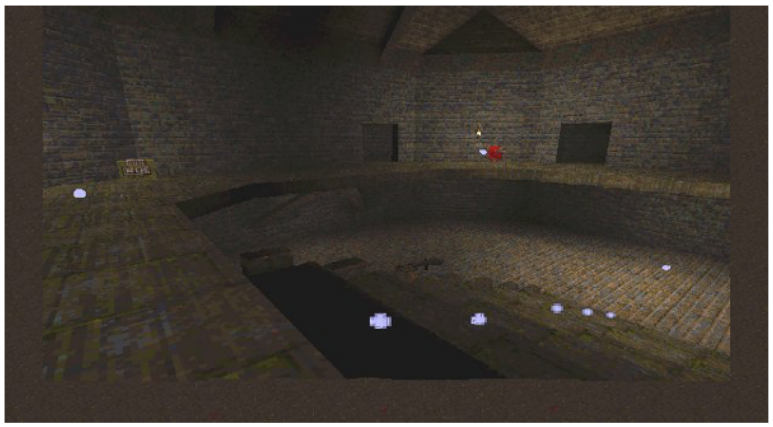
Complete quadtree

https://pdfs.semanticscholar.org/presentation/09ea/138e53fc96e1a6be02505a5cc4cc7d49dda0.pdf

Path Networks

- Does not require the agent to be at one of the path nodes at all times. The agent can be at any point in the terrain allowed by physics simulation.
- When the agent needs to move to a different location and an obstacle is in the way, the agent can move to the nearest path node accessible by straight-line movement and then find a path through the edges of the path network to another path node near to the desired destination.

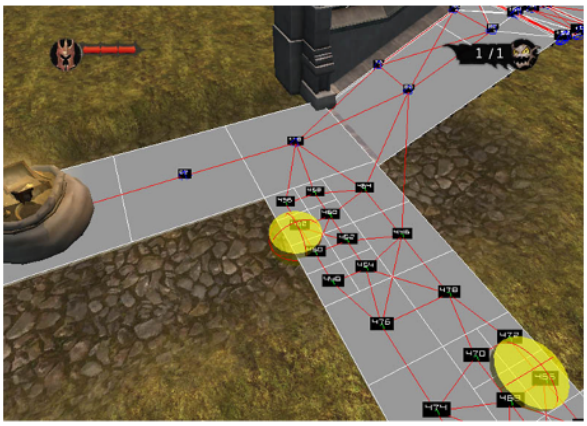
Path Network



7

Reaper bot from Quake

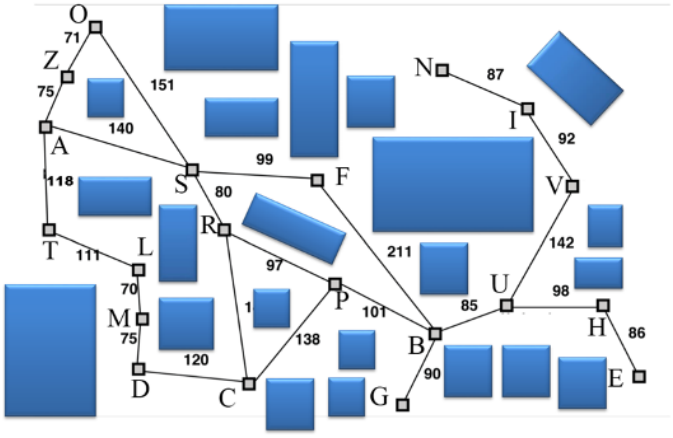
Path Network



Path Networks

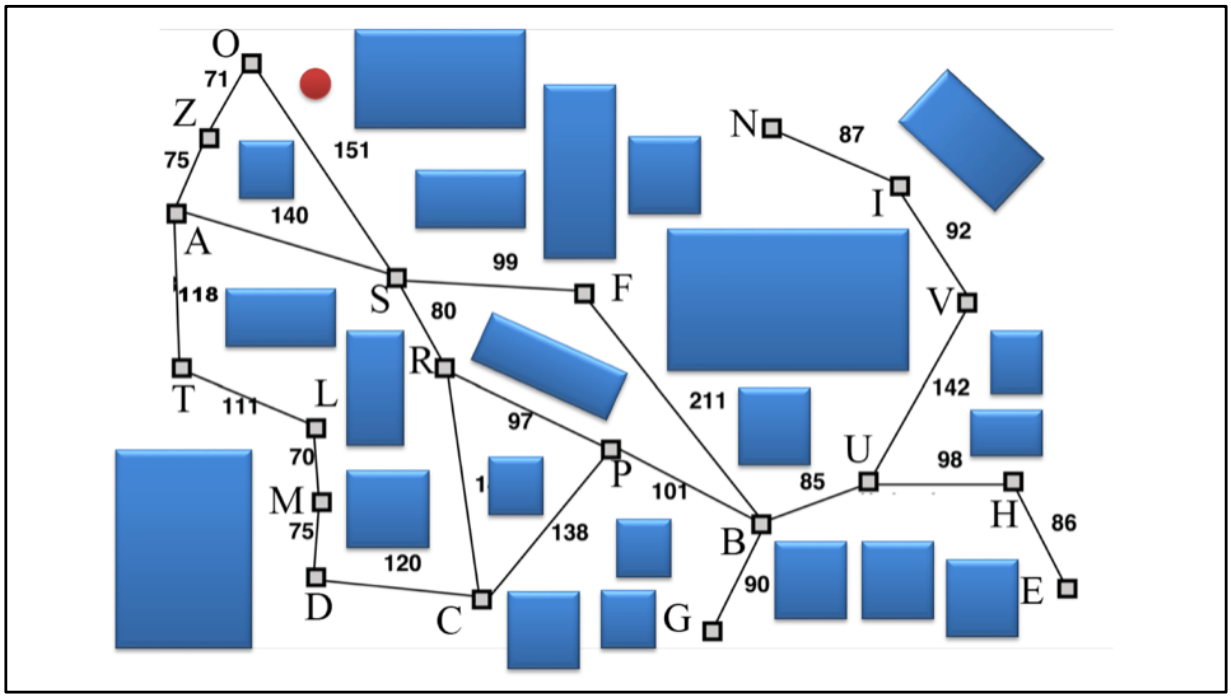
- Discretization of space into sparse network of nodes
- Two-tiered navigation system – Local, continuous
  – Remote
- Connects points *visible to each other* in all important areas of map
  - Facilitated by raycast
- Usually hand-tailored (can use flood-fill or points of visibility)

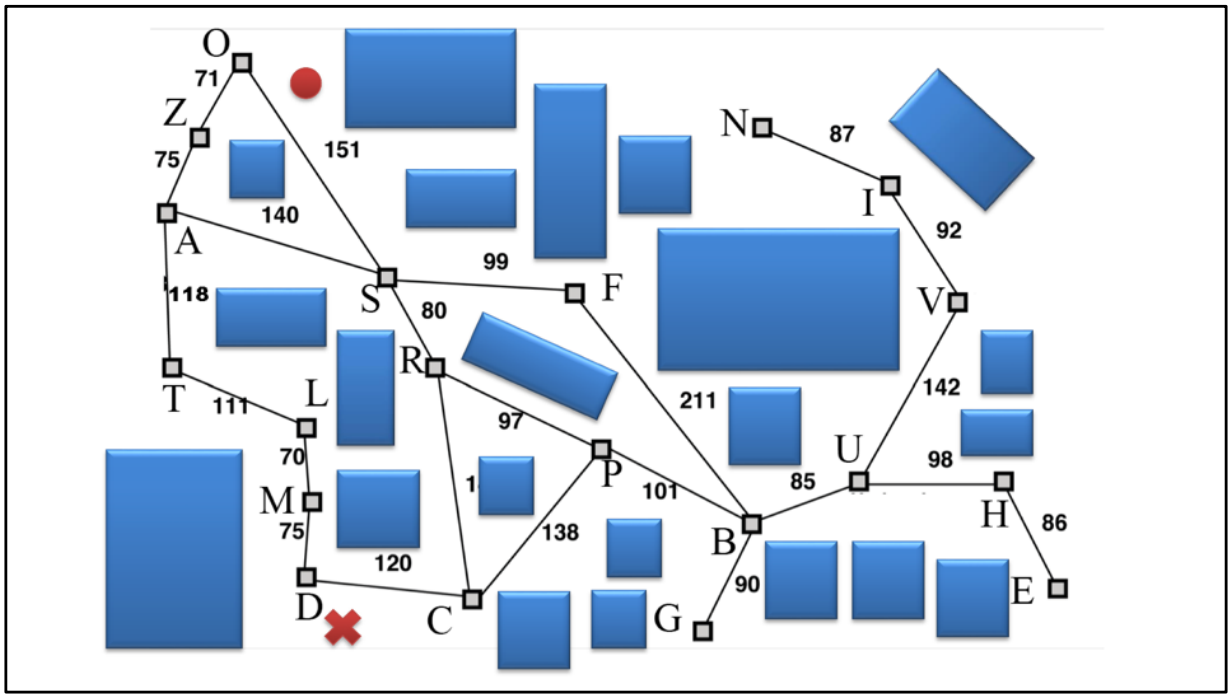
Path Networks

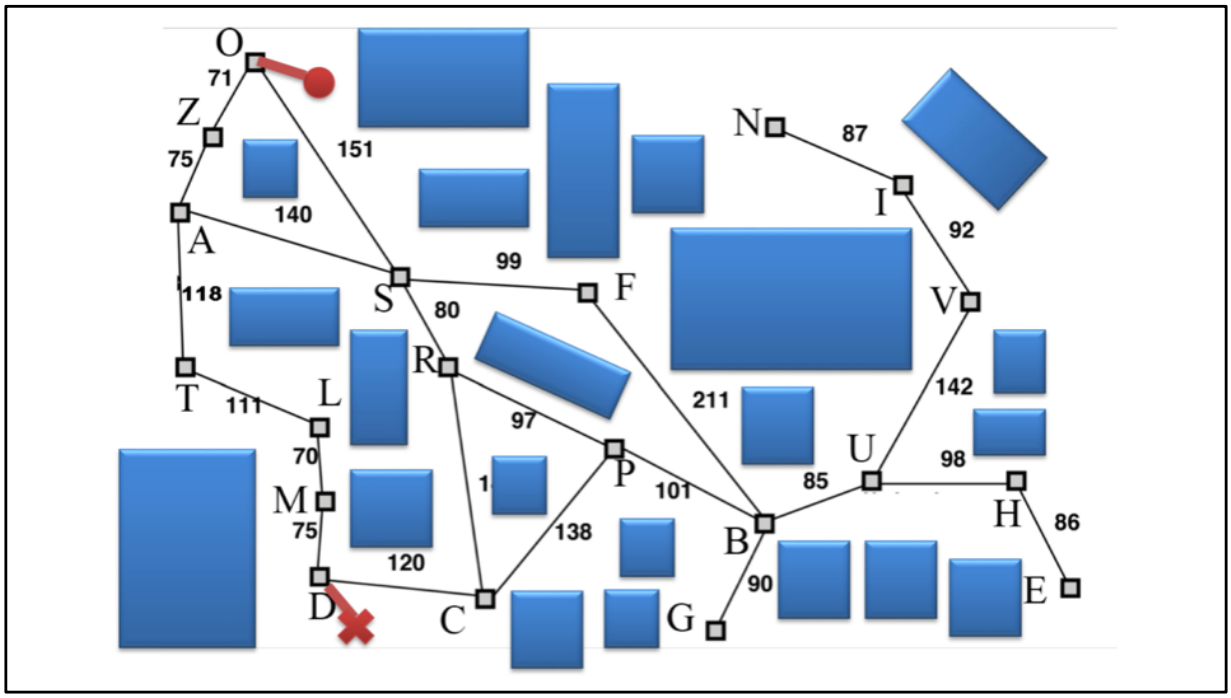


Using the path network

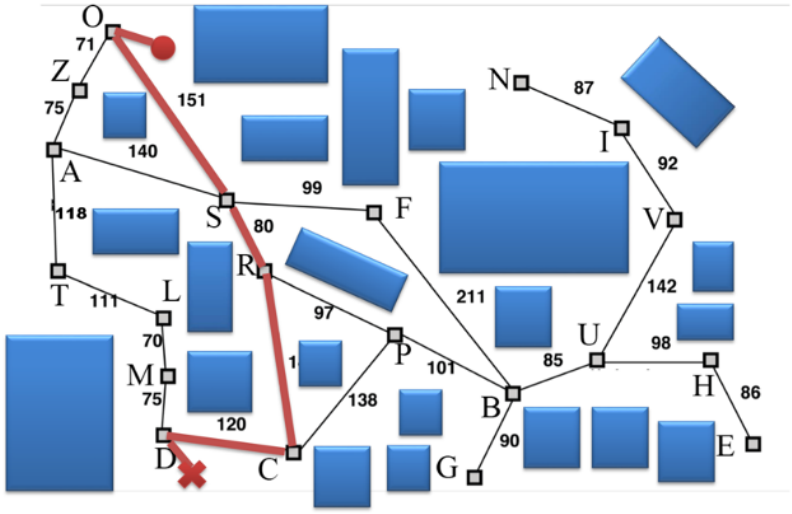
- Basic AI steps when told to go to target X
  - 1. Find the closest visible graph node (A)
  - 2. Find the closest visible graph node to X (B)
  - 3. Search for (lowest cost) path from A to B
  - 4. Move to A
  - 5. Traverse path
  - 6. Move from B to X
- Problem: Unsightly paths.


O
71
Z
75
151
140
A
118
99
S
80
F
N
87
I
92
V
R
97
211
142
T
111
L
70
M
75
120
P
101
85
U
98
H
86
138
B
90
D
C
G
E

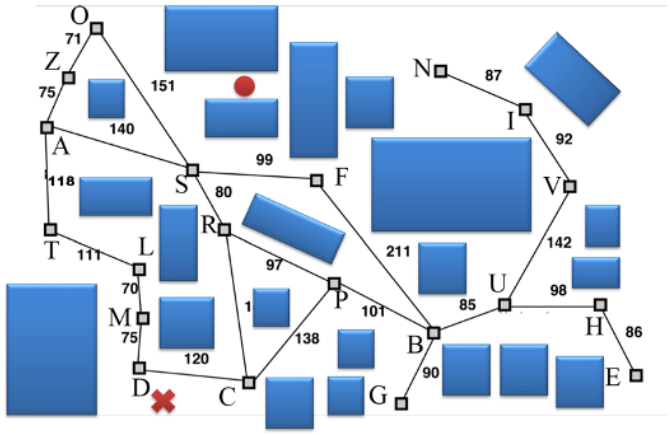

O
71
Z
75
151
140
A
99
118
S
80
F
N
87
I
92
V
R
T
111
L
97
211
142
70
P
101
U
98
M
75
1
85
H
86
120
138
B
D
C
90
G
E


O
71
Z
75
A
140
151
99
S
80
F
118
T
111
L
70
M
75
120
D
C
R
97
P
101
138
G
N
87
I
92
V
142
211
U
85
98
H
86
B
90
E

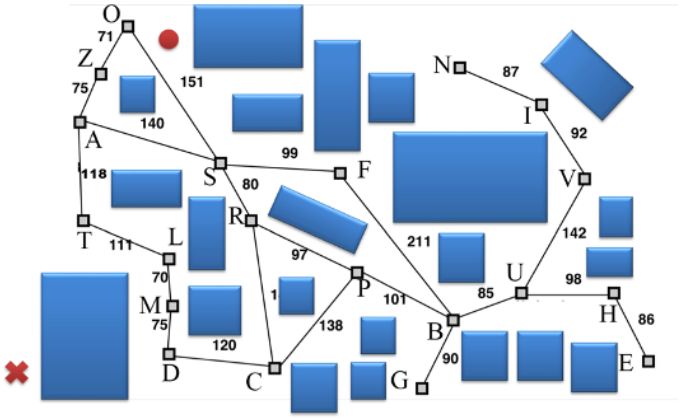
Strange Double-backs (quantization)



Can’t see a navigation point



Destination not in sight of nav point

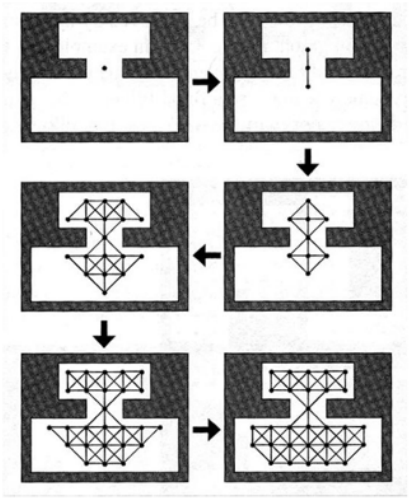


Path Network - Building

- Manual Placement
  - Level designer drops virtual breadcrumbs
- Automatic Placement…

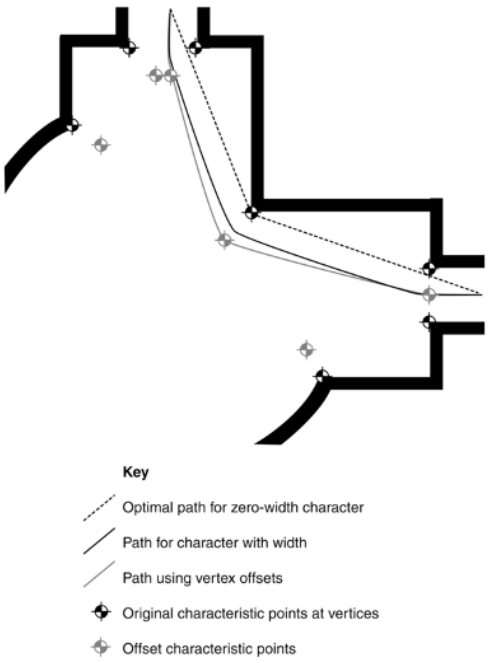
Path Network – Automatic Placement – Flood Fill

- Start with *seed*
- Expand by adding uniformly
- Designer might move, delete, add manually afterwards
- Ensure **all nodes and edges are at least as far from walls as agent's collider radius**



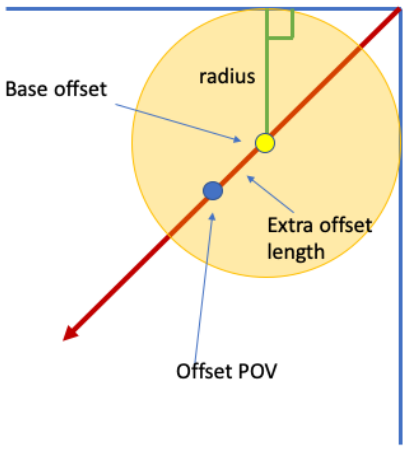
Points of Visibility

- Convex angle obstacle features create natural inflection points for efficient paths through the environment.
- Real world maps needs to be simplified (possibly use colliders) as too many points to be practical
- Inflection points must be offset by some distance to leave room for agent
  - Offset along bisecting line of angle
  - Should to be further than the radius of the agent to allow flexibility/buffer

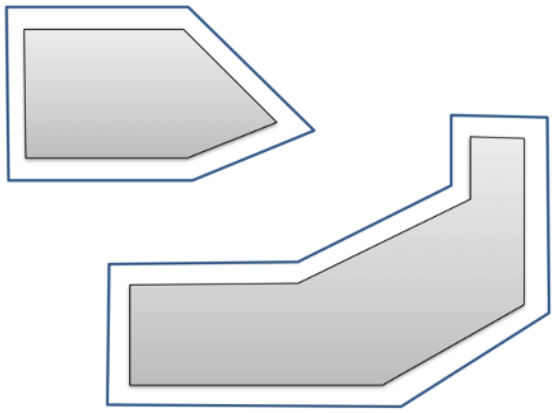


Points of Visibility

- Concave angles can be used as well for better coverage of a navigable space (especially a boundary containing the agent)
- Similar to previous convex angle offsetting, but you additionally need to find point on the angle bisecting line that is at least the agent's radius away from a line that is coincident with one of the two line segments of the convex angle.
  - Add some extra offset as well so agent doesn't get stuck
- Concave POVs are not natural path inflection points, so only useful if you implement path refinement as well

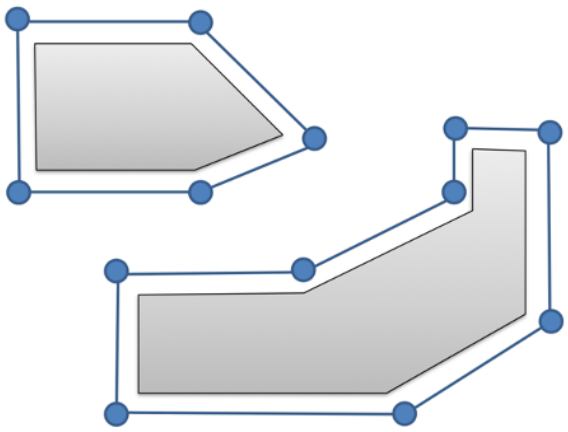


Automate Path Network – POV Graph



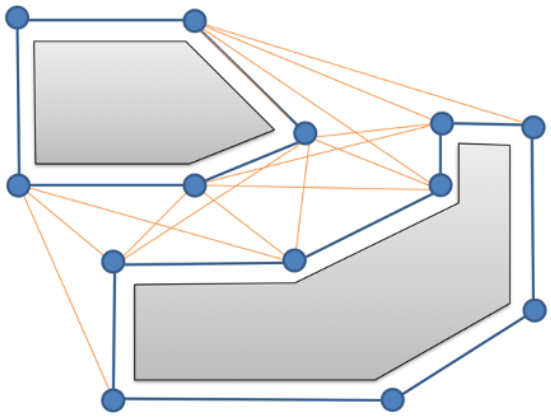
Automate Path Network – POV Graph

- Offset Vertices



Automate Path Network – POV Graph

- Vertices define nodes
- Determine visibility between nodes for edges
- Tends to find natural efficient paths (e.g. agent hugs corners around obstacles)



Points of Visibility

- Can result in graph bloat
- Can end up going down a rabbit hole of adding more and more software features that never quite work
- Often requires a lot of manual tweaks, offsetting the benefits

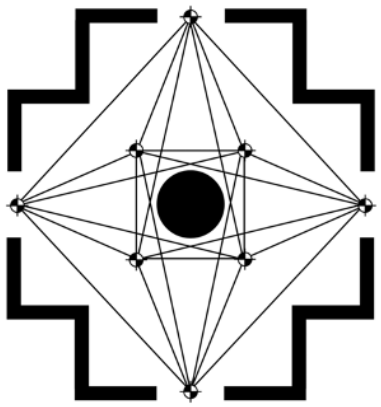
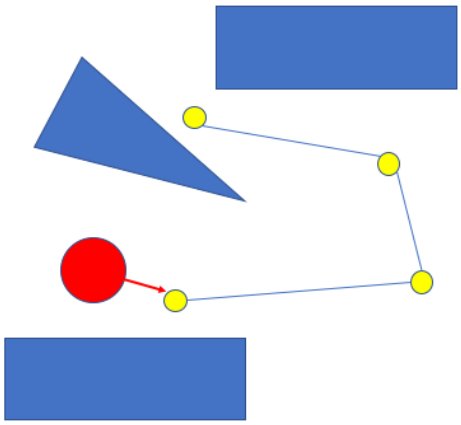


Figure 4.28: Points of visibility graph bloat

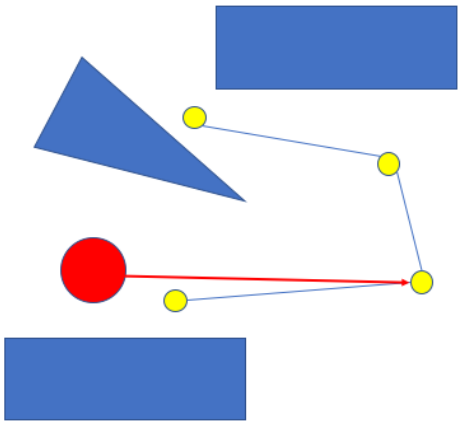
Path Network: Steering

- Improve path by NPC steering/skipping-ahead to furthest node in path that it can see
  - Use raycast, but watch out for pits!
- May clip a corner
  - Cast width-offset parallel rays
- Can also lookahead on edges through iterative subdividing
  - But more expensive



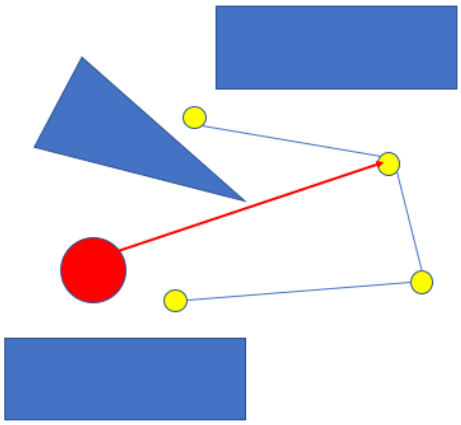
Path Network: Steering

- Improve path by NPC steering/skipping-ahead to furthest node in path that it can see
  - Use raycast, but watch out for pits!
- May clip a corner
  - Cast width-offset parallel rays
- Can also lookahead on edges through iterative subdividing
  - But more expensive



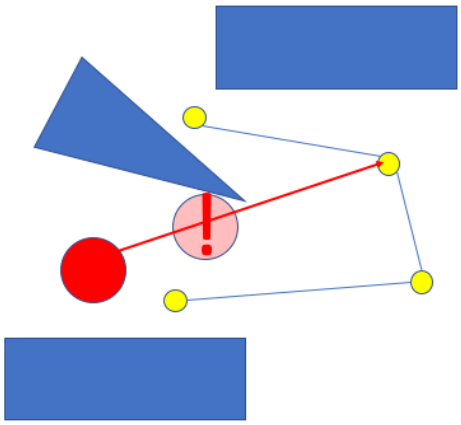
Path Network: Steering

- Improve path by NPC steering/skipping-ahead to furthest node in path that it can see
  - Use raycast, but watch out for pits!
- May clip a corner
  - Cast width-offset parallel rays
- Can also lookahead on edges through iterative subdividing
  - But more expensive



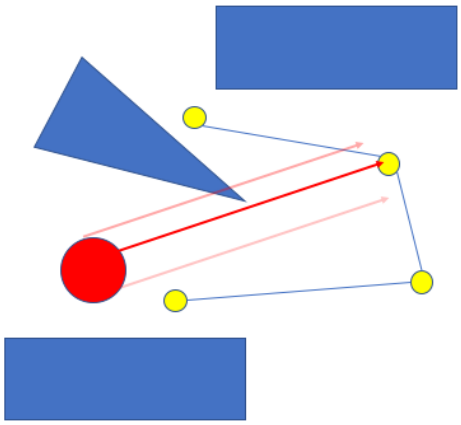
Path Network: Steering

- Improve path by NPC steering/skipping-ahead to furthest node in path that it can see
  - Use raycast, but watch out for pits!
- May clip a corner
  - Cast width-offset parallel rays
- Can also lookahead on edges through iterative subdividing
  - But more expensive



Path Network: Steering

- Improve path by NPC steering/skipping-ahead to furthest node in path that it can see
  - Use raycast, but watch out for pits!
- May clip a corner
  - Cast width-offset parallel rays
  - Or use Expanded Geometry!
- Can also lookahead on edges through iterative subdividing
  - But more expensive

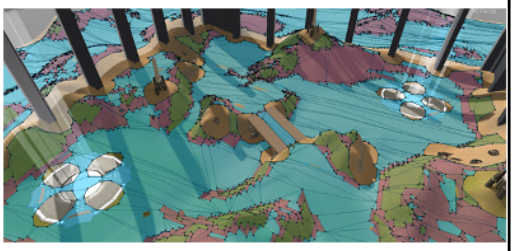


Path Network: Steering Fix - Problems

- Aggressive lookahead can result in problems:
  - Turns too sharp that cause NPC to slow down or..
  - Overshoot if NPC performance envelope doesn't match
- (However, even following a path as it is can be problematic)

Path Network: Steering Fix - Problems

- NPC steering behavior ultimately is not a perfect solution to *fixing* path network problems
- Raycasting will still miss some things
- Raycasting can easily become expensive!
- Especially problematic on variable height terrain and different terrain types
- NPC may take a shortcut but end up walking in slow terrain instead of staying on the path of fast terrain (e.g. mud versus road)



Path Network - Pros

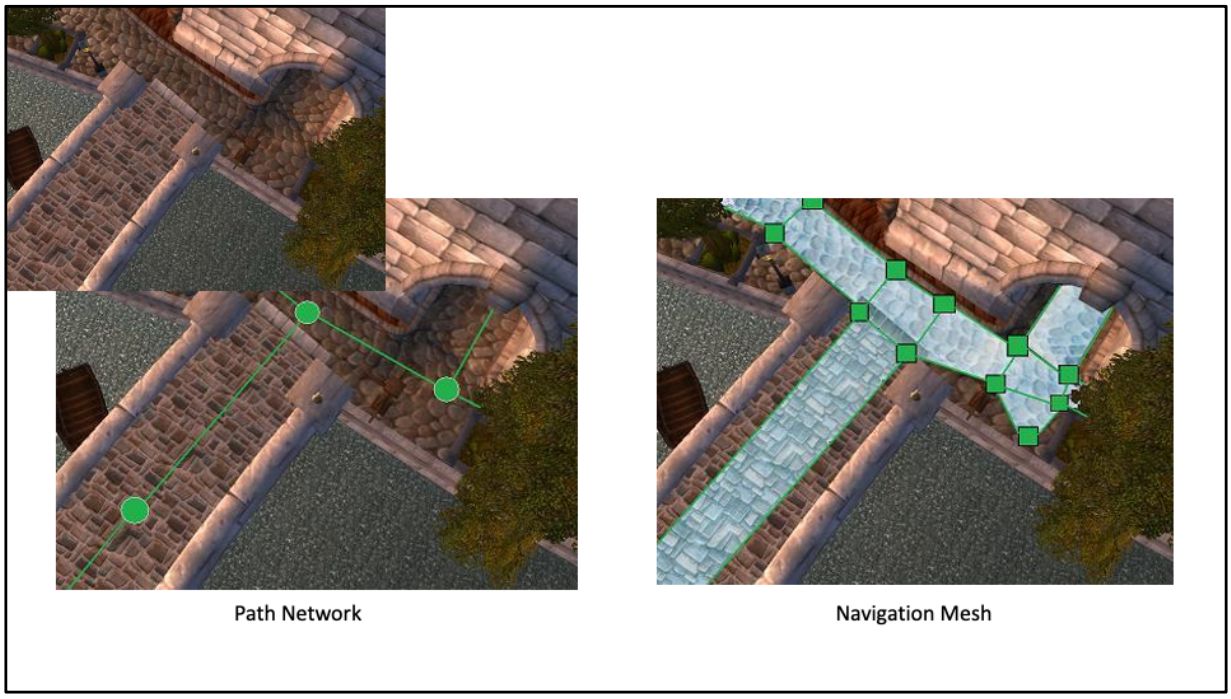
- Discretization of space is (can be) very small
- Does not require agent to be at one of path nodes at all times (unlike grid)
- Continuous, non-grid movement in local area
- Switch between local and remote navigation
- Plays nice with “steering” behaviors (we will discuss these later)
- Good for FPS, RPGs
- Can indicate special spots (e.g. sniping, crouching, etc.)

- Discretization of space can be smaller
- Continuous, non-grid movement in local area
- Can work with auto map generation
- Switch between local and remote navigation
- Can play nice with steering behaviors

Path Network - Cons

- Valid NPC positions might not be able to see a node in the network
- Jagged path shape
- Dynamic and rolling terrain issues
- Storage/complexity of network to get good coverage and good path shape (esp. auto-gen)
- NPC going off the network path can be dangerous (getting stuck, etc.)

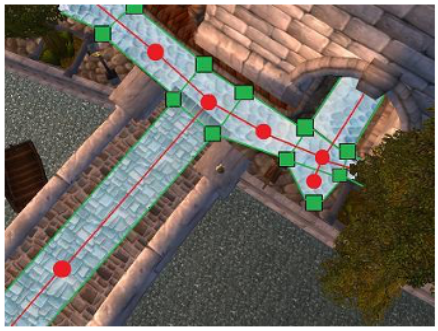
Dynamic terrain issues
Resources (potentially lots of nodes)
    May need to use simplified geometry, perhaps colliders rather than graphic representation
More difficult with rolling terrain



http://www.cs.uu.nl/docs/vakken/mpp/other/path_planning_fails.pdf

NavMesh

- Defines navigable polygonal areas with convex regions
- Adjacent edges define graph connectivity



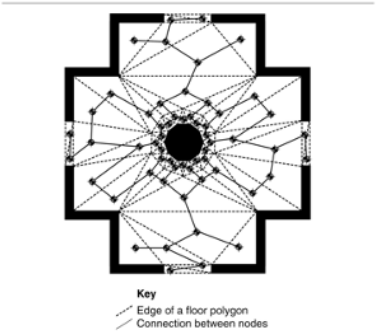


Figure 4.29: Polygonal mesh graph

Blue areas with green edges are NavMesh

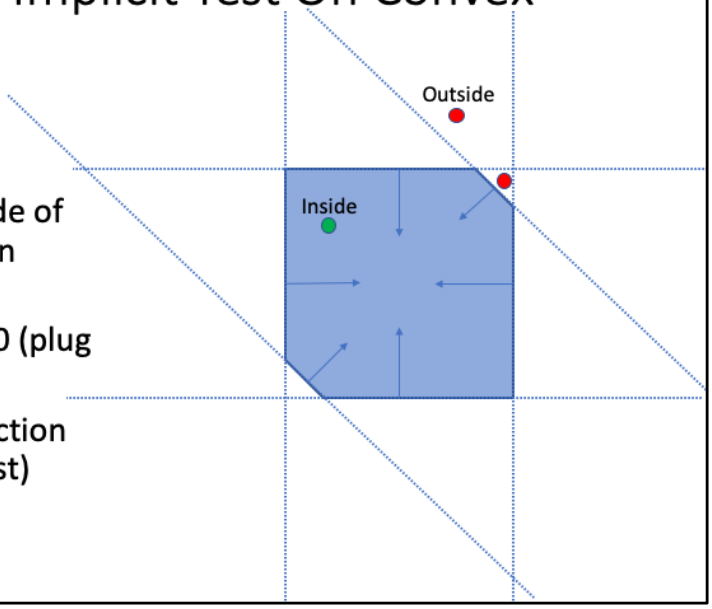
Red denotes the graph (that can be searched) which is defined by the NavMesh

Convexity

- By definition: convex polygons have all internal angles of less than 180 degrees
  - all diagonals are contained within the polygon
  - a line drawn through a convex polygon in any direction will intersect at exactly two points (meaning agent has nothing to collide with!)
- Can also use implicit test

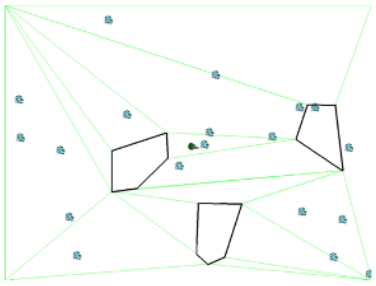
Point Containment Implicit Test On Convex Polygon

- A convex polygon is the intersection of half-planes
- Point must be on correct side of each line defined by polygon edges (line segments)
- Implicit test: ax + by + c >= 0 (plug in test point, check sign)
- Consistent CW or CCW direction around poly (affects sign test)



Can also do test of point P with each edge AB via dot product against AP

Generating the Mesh



For point a in world points:
  For point b in world points:
    For point c in world points:
      if (it is a valid triangle) and !exists:
        add triangle to mesh

Valid Triangle:
Cannot cross existing triangle!

Iterate through triangles to merge to quads
Iterate through quads to merge to 5-sided shapes...

Greedy Mesh Generation

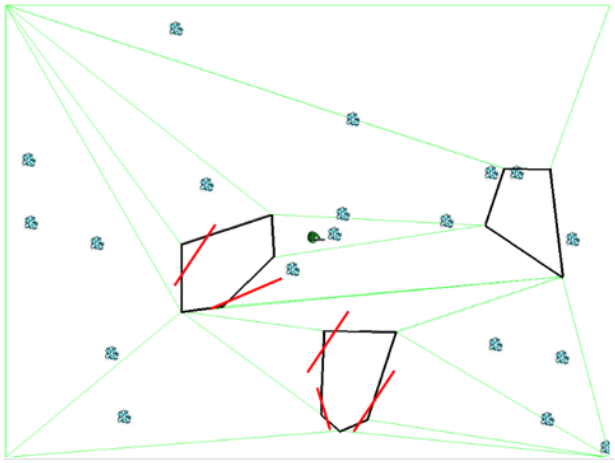
- First, find all triangles:
  - Pick a point on an obstacle, or boundary
  - Successor points are:
    - Along edge of common obstacle ***or***
    - Through world space (another obstacle)
  - Pick two successor points for candidate triangle
  - See if they are successors of each other
  - Cannot cross an existing triangle
  - Repeat until you can't make more triangles from original point
  - Pick a new starting point

Greedy Mesh Generation

- Find high-order polygons:
- For any 2 triangles that share an edge, check if merged polygon is still convex
- If so, merge
- Repeat merging progressively higher-order polygon merge attempts as above

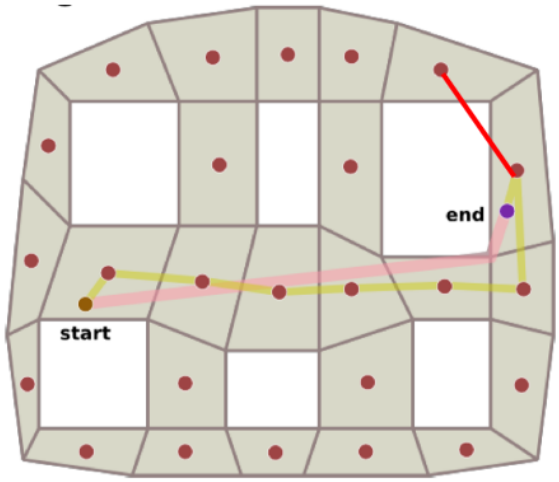
Generating the Mesh

- Don't join verts to triangles if intrudes on colliders



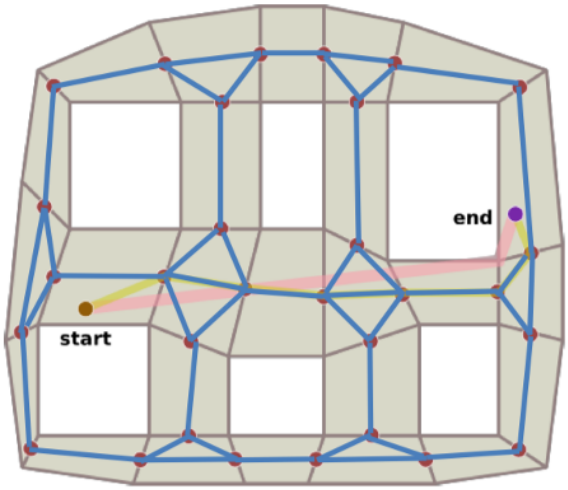
Waypoints from NavMesh

- Put a waypoint in center of each navmesh poly
  - Important to get a good set of navmesh polys



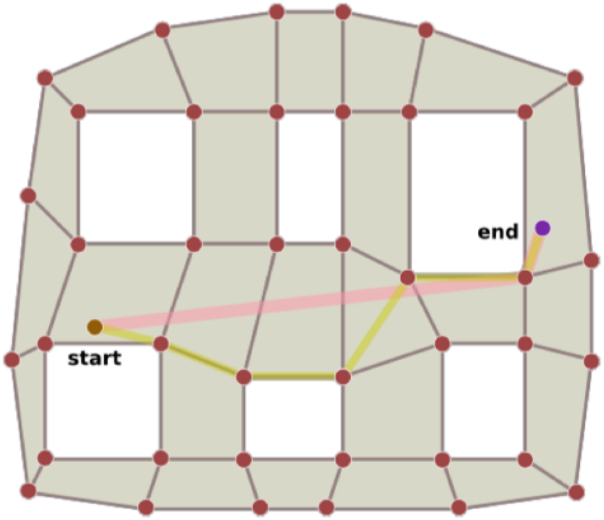
Waypoints from NavMesh

- Put a waypoint at center of adjoining edges



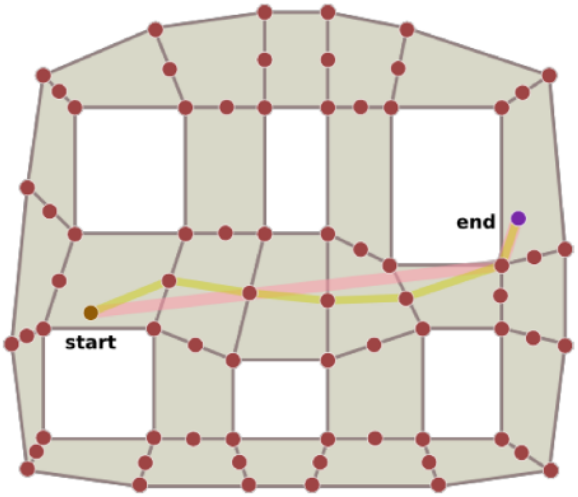
Waypoints from NavMesh

- Put a waypoint at corner of each obstacle

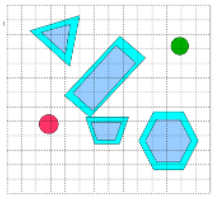


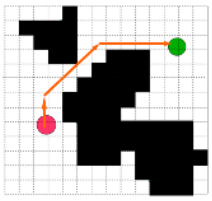
Waypoints from NavMesh

- Put a waypoint at edges and corners

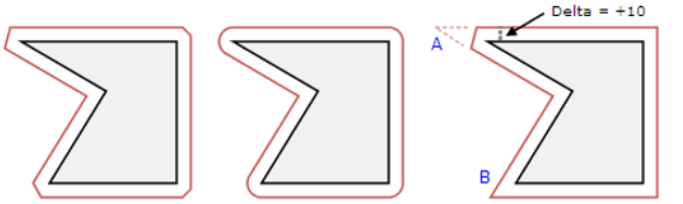


Expanded Geometry





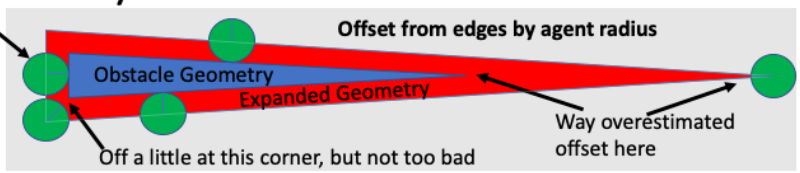
- Enlarge obstacles so that discretized space takes into account NPC size (radius)
- Easier to bake this into nav structures rather than continuously calculate
- Applicable to all types of nav structures!

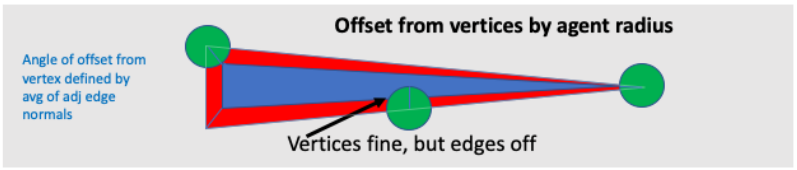


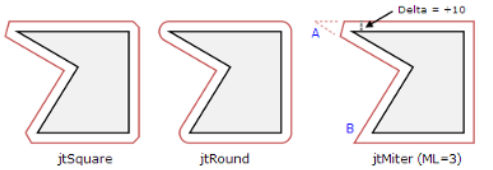
The angle at vertex 'A' is more acute than that at 'B' and, since a mitered offset would exceed the specified limit (3 x delta), its offsetting is 'squared'.

Expanded Geometry – The Issue of Corners

- Expanding edges can result overestimated offsets
- Expanding vertices can result in underestimated offsets
- Equidistant expansion introduces non linear curvature (curved at corner offsets)
- Squaring off/selective mitering is compromise to avoid curves
- Also consider: tesselate curve





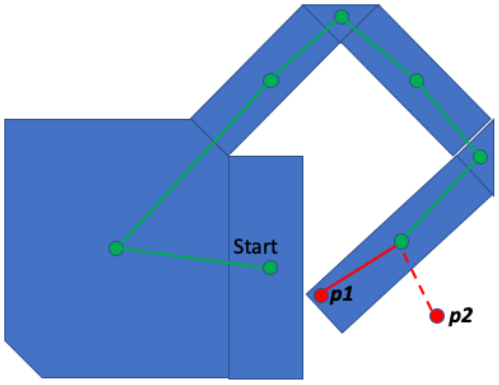


Expanded Geometry

- Very well suited to improving Navmeshes, but can work with grids and path networks
- Similarities to process of generating POV path network
- Once you have employed expanded geometry, you can consider the agent to be a single point. This greatly simplifies tests to see where an agent can go on the navmesh (or other discretized space). A single raycast is sufficient.

Find if (and where) arbitrary point is on a NavMesh

- Can use A* and geometry intersection test with navmesh convex poly (but probably not best way)
- Start at arbitrary NavMesh node
- Use distance to candidate point (***p***) as heuristic
  - We will assume a virtual node on the graph at ***p*** connected to node of navmesh poly it intersects with
  - Test point containment at each node's navmesh poly (convex)
    - Optionally store navmesh poly bounding radius at each node for quick first pass containment filtering
- Worst case, all nodes searched
- **Probably better solution: use bin lattice, quadtree, octree, etc., to organize navmesh polys for quick search**



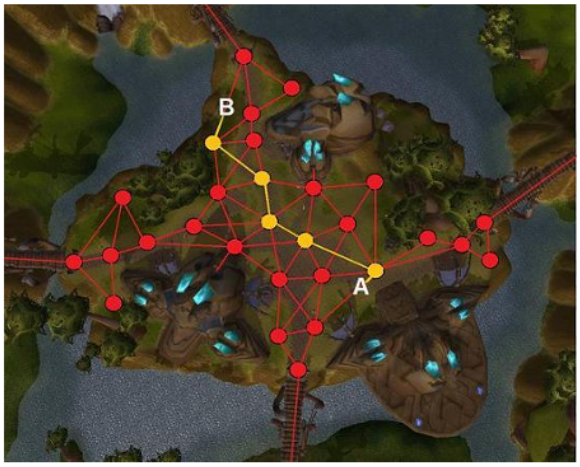
Placing destinations on NavMesh

- Best to make sure candidate destinations are on NavMesh from the start of the simulation
- Consider an automated build/bake/audit process
- Use spawn points
- Track non-agent moving objects (such as player) relying on coherence (assumption that object moves from one adjacent navmesh poly to another)

Path Network

NavMesh

Path Network – Jagged Path



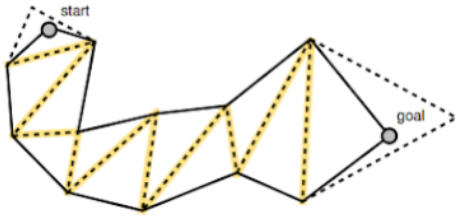
Zig zag path is undesirable

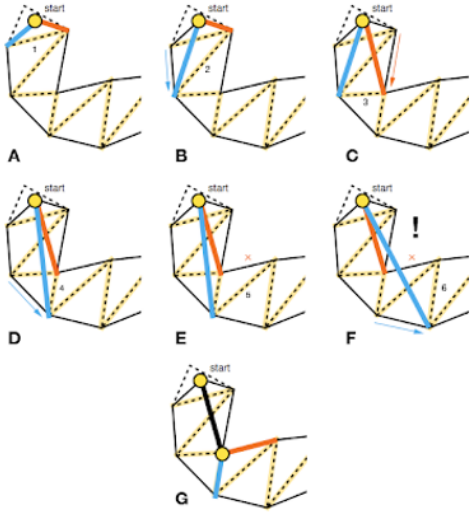
NavMesh – Path

We can safely straighten out things because the NavMesh tells us which areas are traversable

To accomplish similar results with a Path Network would require a very large amount of nodes!

Simple Stupid Funnel Algorithm





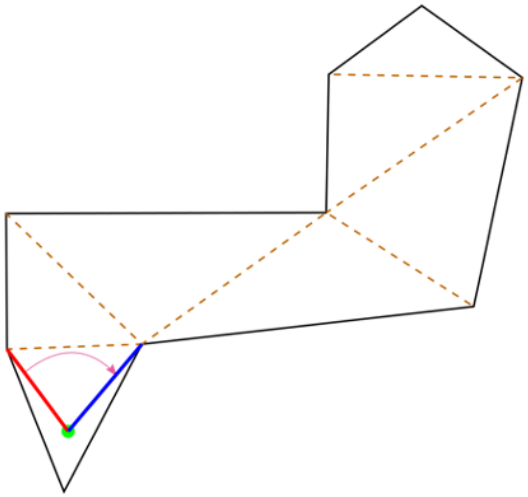
This version works well with expanded geometry navmeshes

From GDC18 Horizon Zero Dawn presentation

http://digestingduck.blogspot.com/2010/03/simple-stupid-funnel-algorithm.html

Horizon Zero Dawn version of Funnel

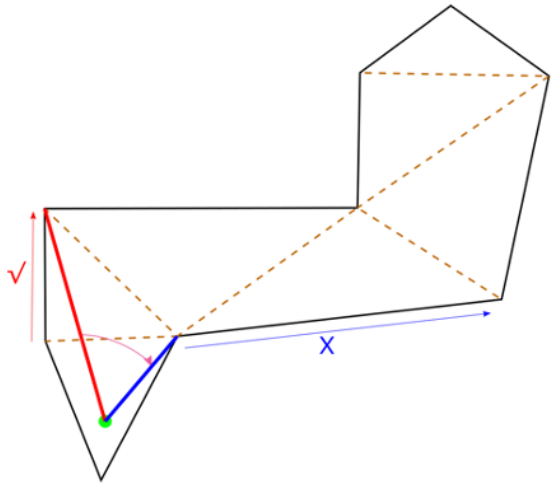
- This version of funnel algorithm works well if you don't have expanded geometry
- Create a funnel from starting position to portal points



We start with our **planned polygon path**. Starting at the start position of the character, we create **a funnel** towards the first points that **form the portal** to the next polyon.

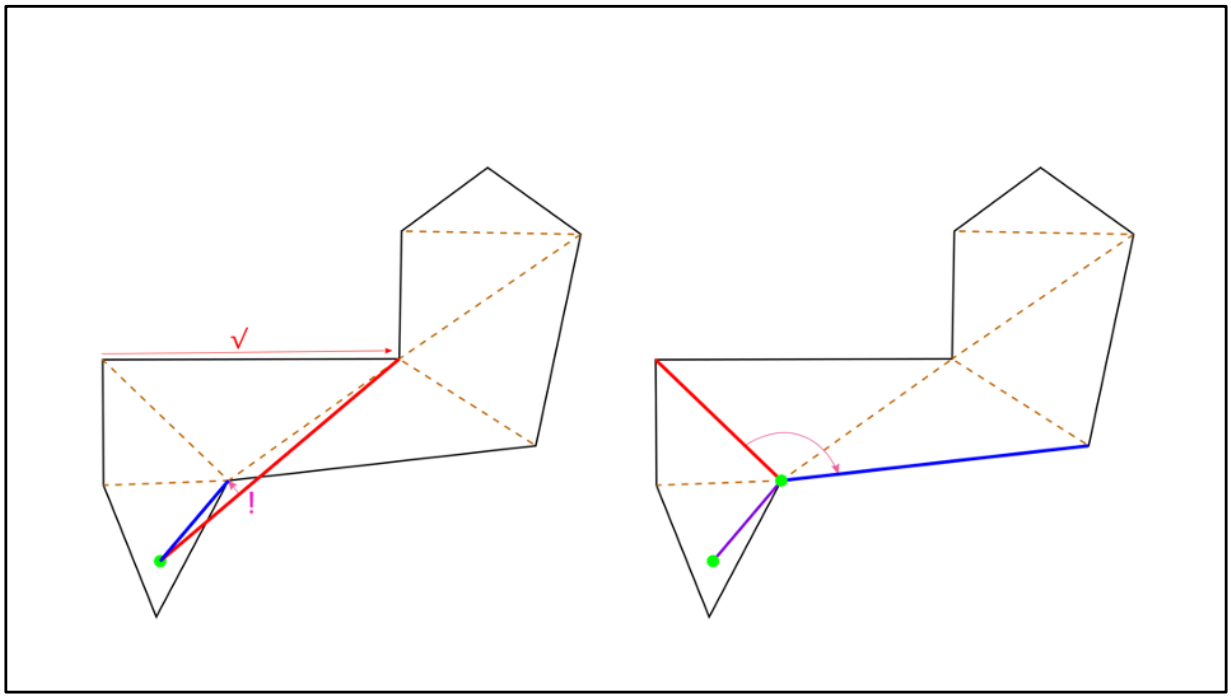
From GDC18 Horizon Zero Dawn presentation

- Check next 2 points, Set funnel side to each point if inside funnel



We then look at **each following portal points**
and **shrink** the funnel each time one of the portal points **lie within** that funnel

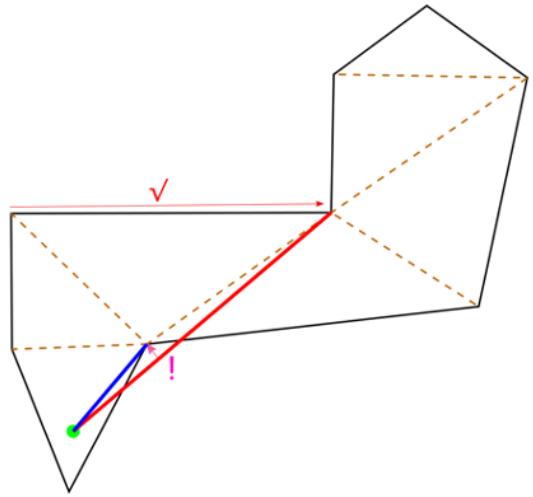
From GDC18 Horizon Zero Dawn presentation



As soon one side of the funnel **crosses** the other one, we add the **other funnel point** as a **point in our final point path** and **restart** the process from there

From GDC18 Horizon Zero Dawn presentation

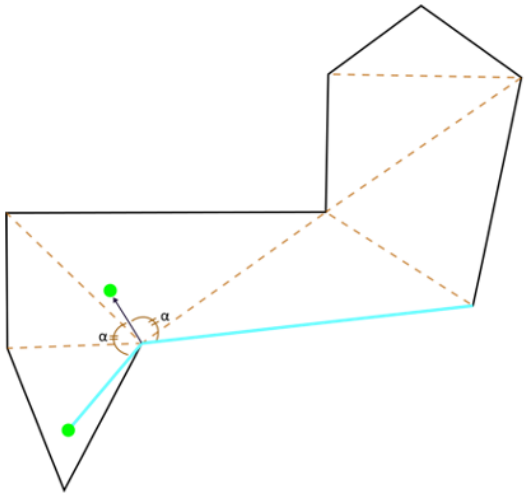
- Starts out the same way, until one side crosses the other



Now our adjusted algorithm **works the same** right until we get here in the algorithm. Because when one side of the funnel crosses the other we don't just add that point to the path. Instead, we first **add an offset** in the **combined direction from the start of the funnel to the portal point and the next wall**. Like so

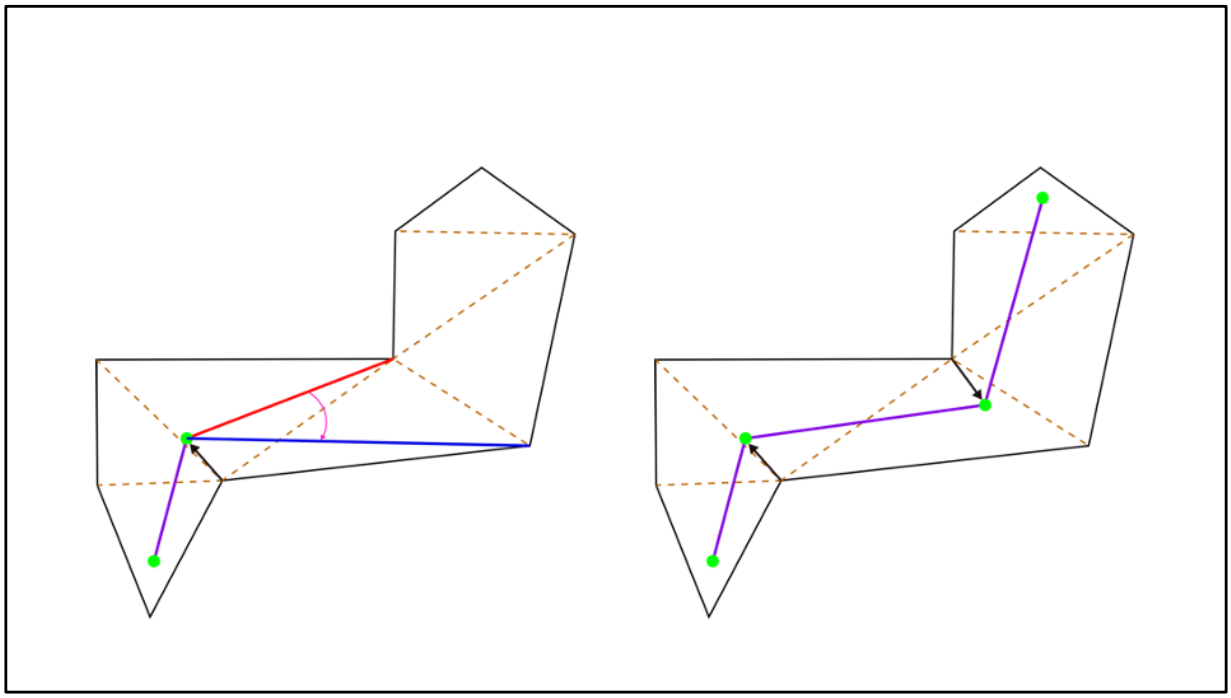
From GDC18 Horizon Zero Dawn presentation

- Distance is approximated based on the default turn circle and wall angle
- Offset is bounded by opposite wall



The distance of the offset is an **approximation** based on **the change of direction of the those vectors shown in blue**, and **the standerd turn circle of the character** traversing the path.
[CLICK]
This offset is **clamped** against the **opposite walls**,
[CLICK]
so that the final point path will always be **inside** the originally planned poly path

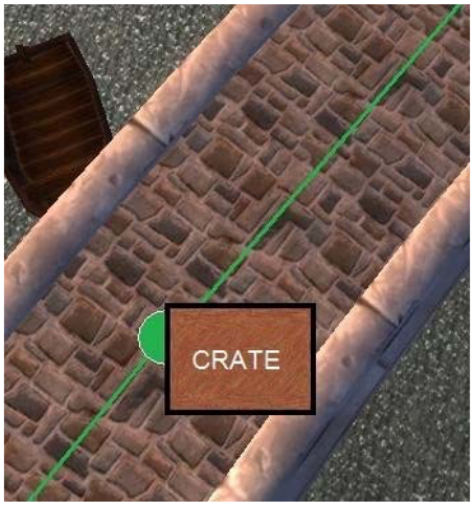
From GDC18 Horizon Zero Dawn presentation



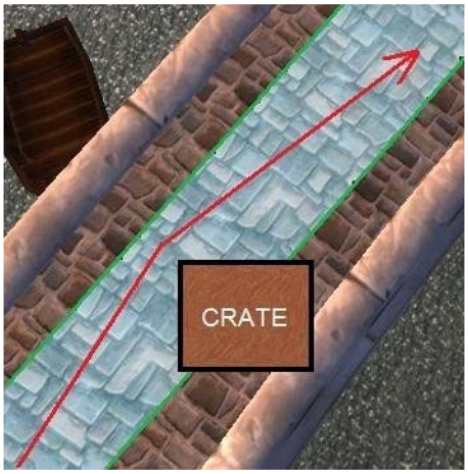
And the **rest is the same**.
The adjusted point with offset is added to the **final point path**.
And the funnel **restarts from there**. The resulting point path does **not hug the walls** anymore, allowing the character traversing the path to **cut the corners**.

From GDC18 Horizon Zero Dawn presentation

Path Network – Unexpected Obstacle – What to do?

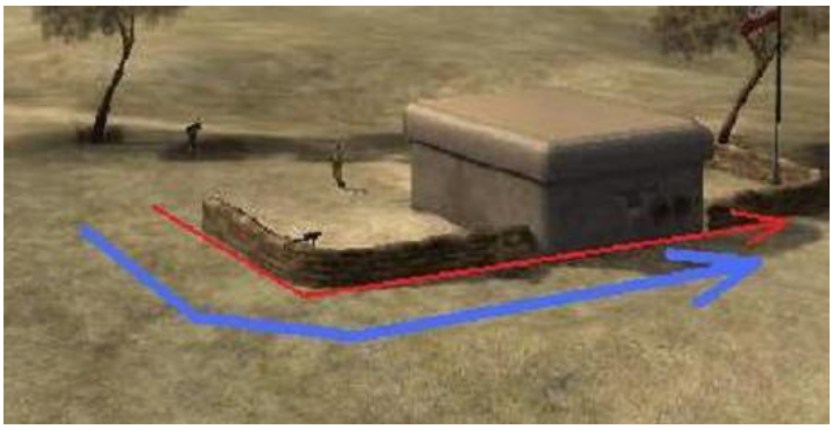


NavMesh – Unexpected Obstacle - Easy



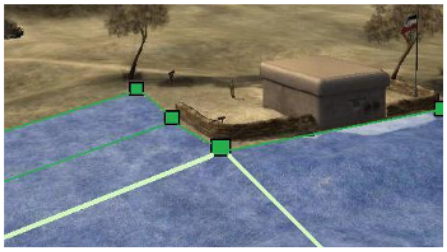
Different Unit Types

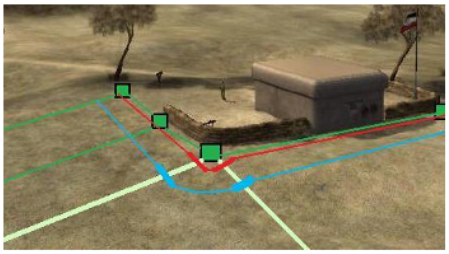
- Tank
- Infantry



NavMesh: Different Unit Types

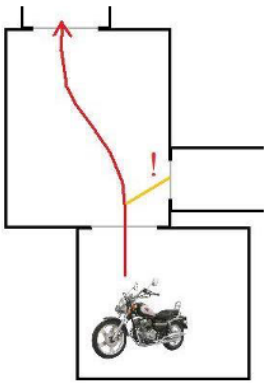
- Relatively easy to offset from edges to compensate for differing radiuses
- (NavMesh can have a default offset already baked in)





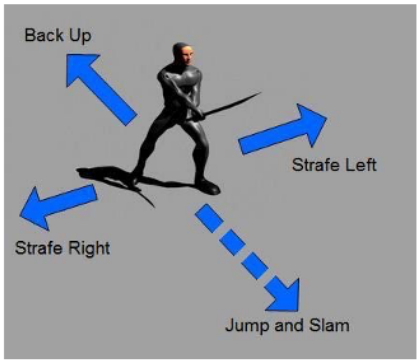
NavMesh: Vehicle Performance Envelope

- Path planning with a NavMesh can more easily consider turn angles and distances, rejecting paths that don't match performance abilities and/or targets



NavMesh – Animated Character Actions

- Can use NavMesh to decide if enough room for particular NPC actions to be executed



NavMesh – Multiple Levels

- Can easily support multiple levels
- Can store *max height* in each NavMesh polygon