WPI

IMGD 4000
Technical Game Development II
Advanced Pathfinding

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Human Interaction in Virtual Environments (HIVE) Lab
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu

A* Pathfinding Search

- Covered in detail in IMGD 3000
- See pseudo-code and links to reference code at

http://web.cs.wpi.edu/~gogo/courses/imgd3000_2011c/slides/imgd3000_08_AI_A_Star.pdf

- **Basic A*** is what you should use for Ghoulie movement (if you choose that option)

WPI

Practical Path Planning

- Just raw A* is often not enough
- Also need:
  - Navigation graphs
    - points of visibility (POV)
    - Navigation mesh (NavMesh)
  - Path smoothing
  - Compute-time optimizations
  - Hierarchical pathfinding
  - Special case methods

Basic Navigation Graph Construction (cont.)

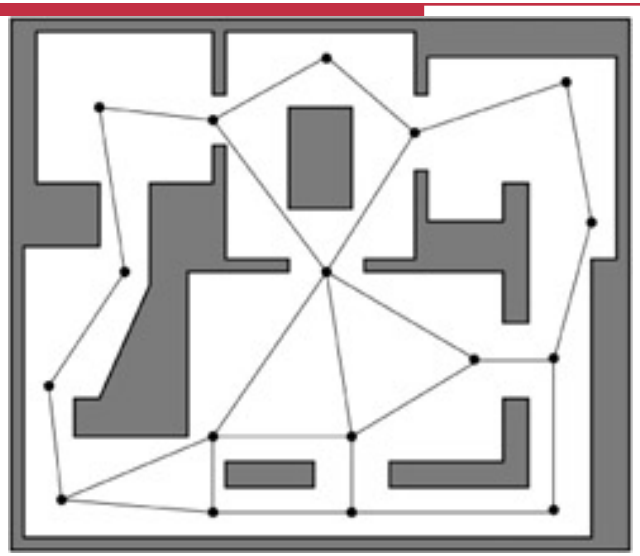
WPI

- Downside:
  - Modest 100x100 cell map has 10,000 nodes and 78,000 edges
  - Can burden CPU and memory, especially if multiple AI's calling in

*Rest of lecture is a **survey** about how to do better...*

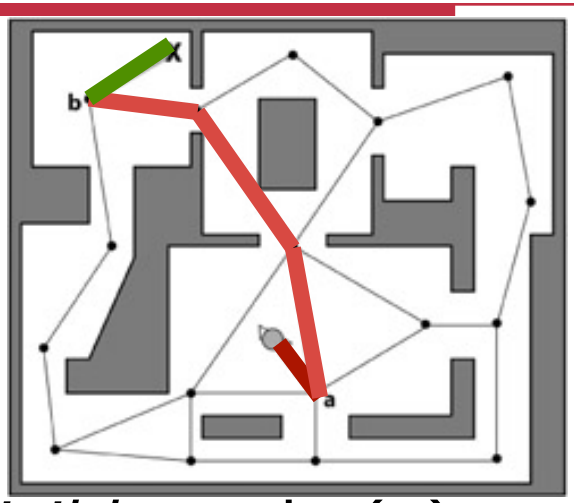
Point of Visibility (POV) Navigation Graph

WPI



- Place graph nodes (usually by hand) at important points in environment, such that *each node has line of sight to at least one other node*

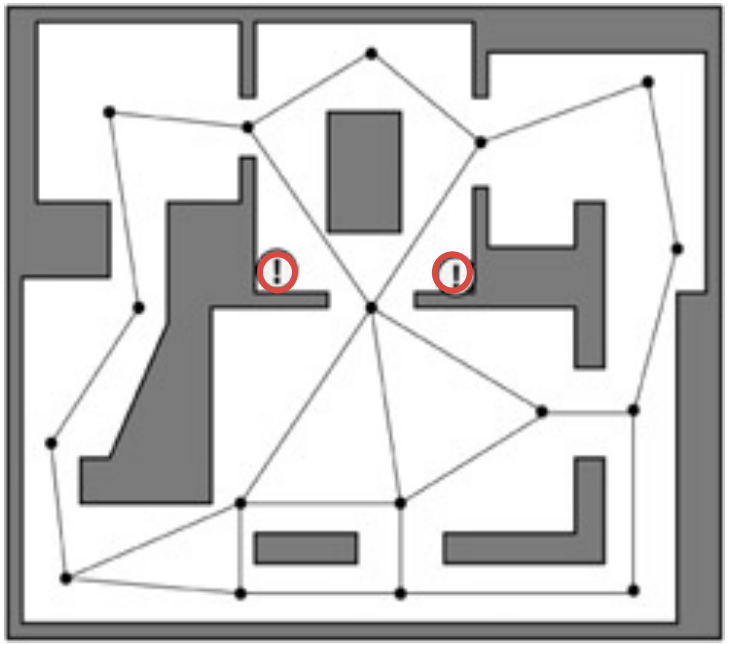
POV Navigation



- Find closest *visible* node (a) to current location
- Find closest *visible* node (b) to target location
- Search for least cost path from (a) to (b), e.g., A*
- Move to (a)
- Follow path to (b)
- Move to target location

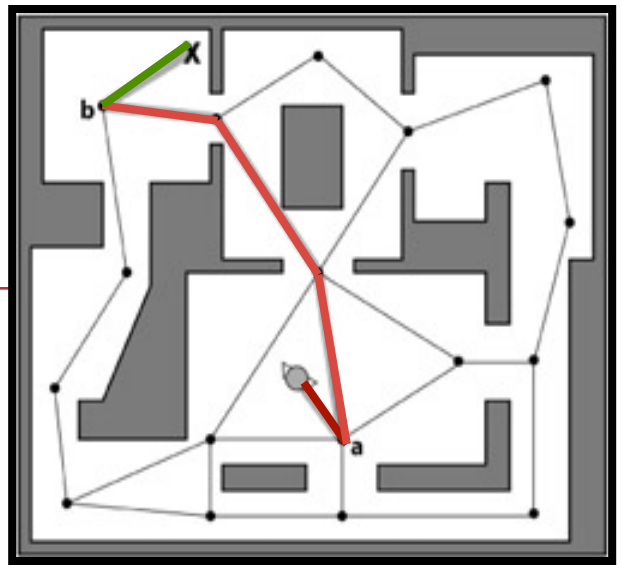
note "backtracking"

Blind Spots in POV



- No POV point is visible from red spots!
- Easy to fix manually in small graphs
- A problem in larger graphs

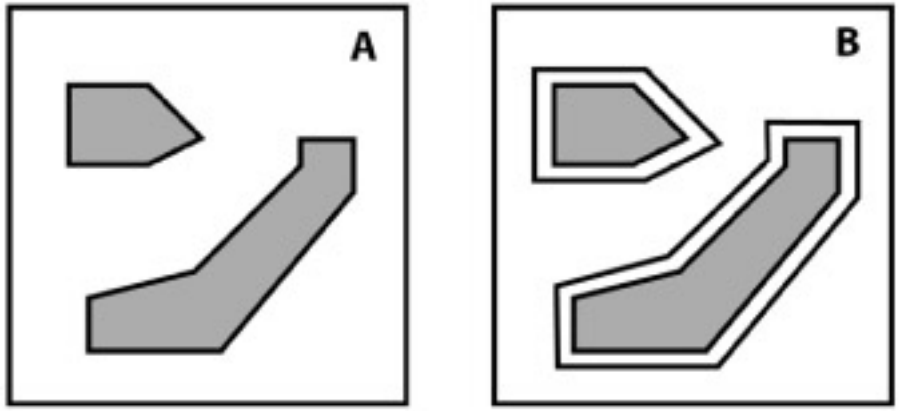
POV Navigation



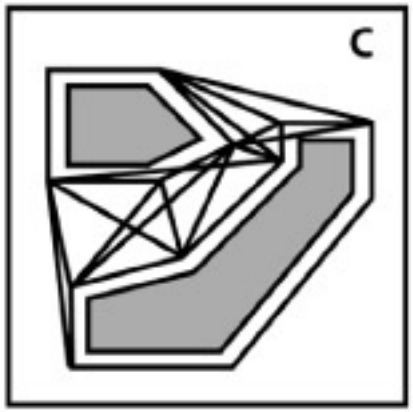
- Advantage
  - Obvious how to build and expand
- Disadvantages
  - Can take a lot of developer time, especially if design is rapidly evolving
  - Problematic if random or user generated maps
  - Can have "blind spots"
  - Can have "jerky" (backtracking) paths
- Solutions
  1. Automatically generate POV graphs
  2. Make finer grained graphs
  3. Path smoothing

Automatic POV by Expanded Geometry

1. Expand geometry by amount proportional to bounding radius of agents
2. Connect all vertices
3. Prune non-line-of-sight points



Simple Geometry Expanded Geometry



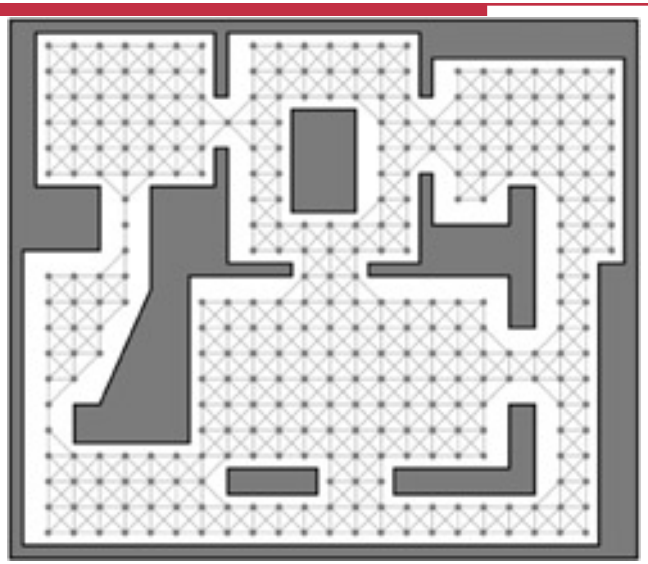
The finished POV graph

NavMesh

- Partition open space into a network of *convex* polygons
  - Why convex?
  - Guaranteed to be path from any point to any point inside
- Very efficient to search
- Can be automatically generated from arbitrary polygons
- Becoming very popular

WPI

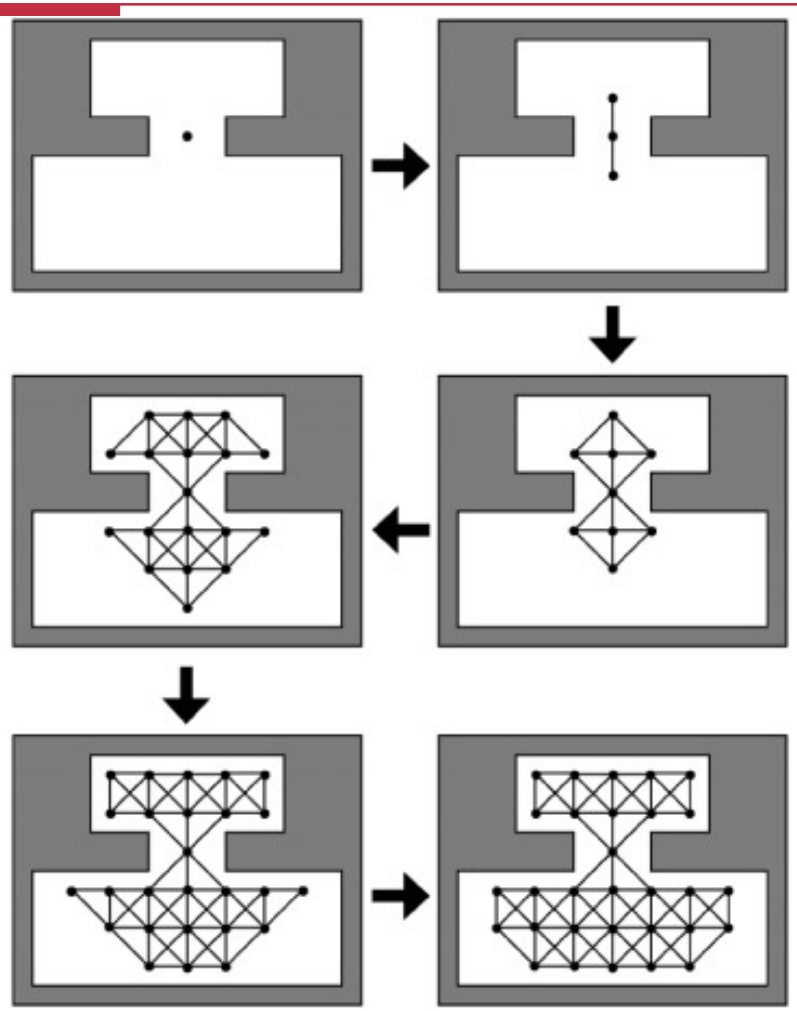
Finely Grained Graphs



- Improves blind spots and path smoothness
- Typically generate automatically using "flood fill"
- Back to similar performance issues as tiled graphs

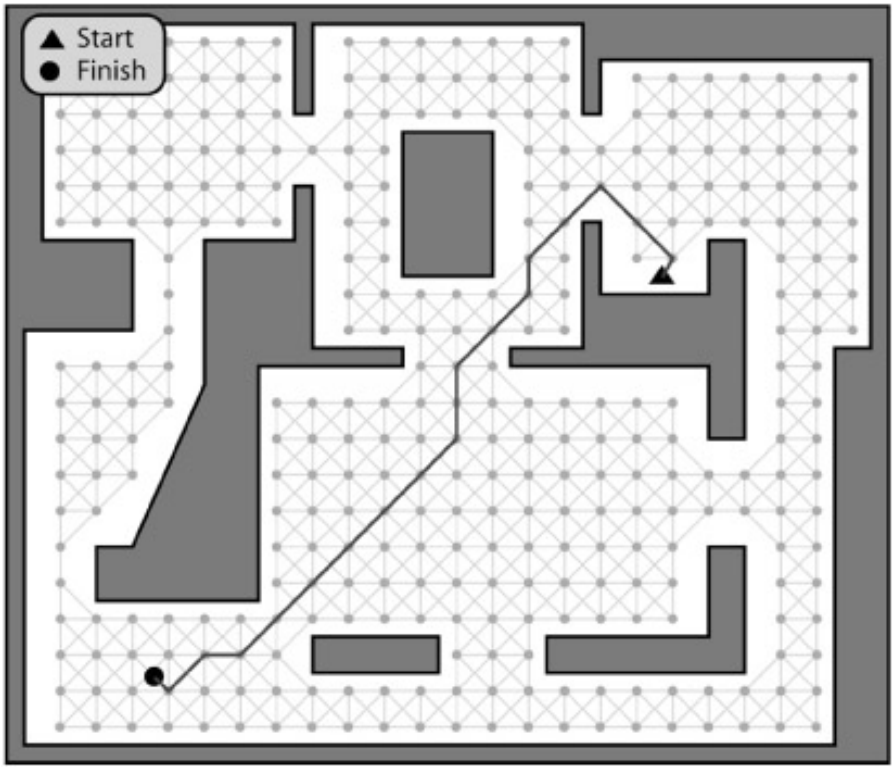
Flood Fill

- Same algorithm as in "paint" programs



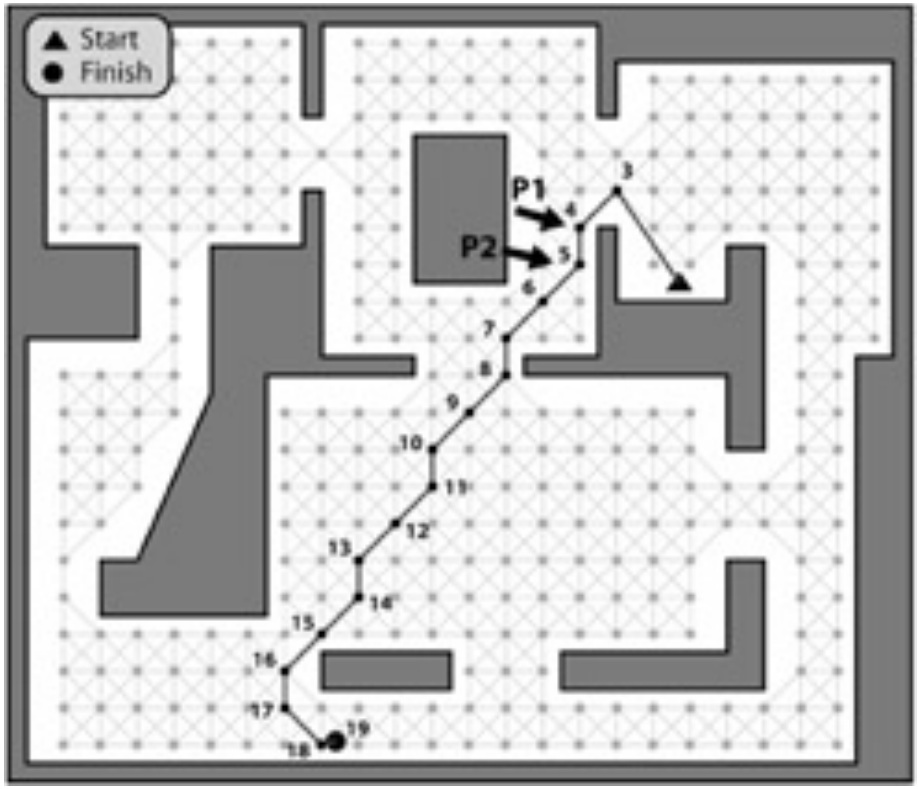
Path Finding in Finely Grained Graph

WPI



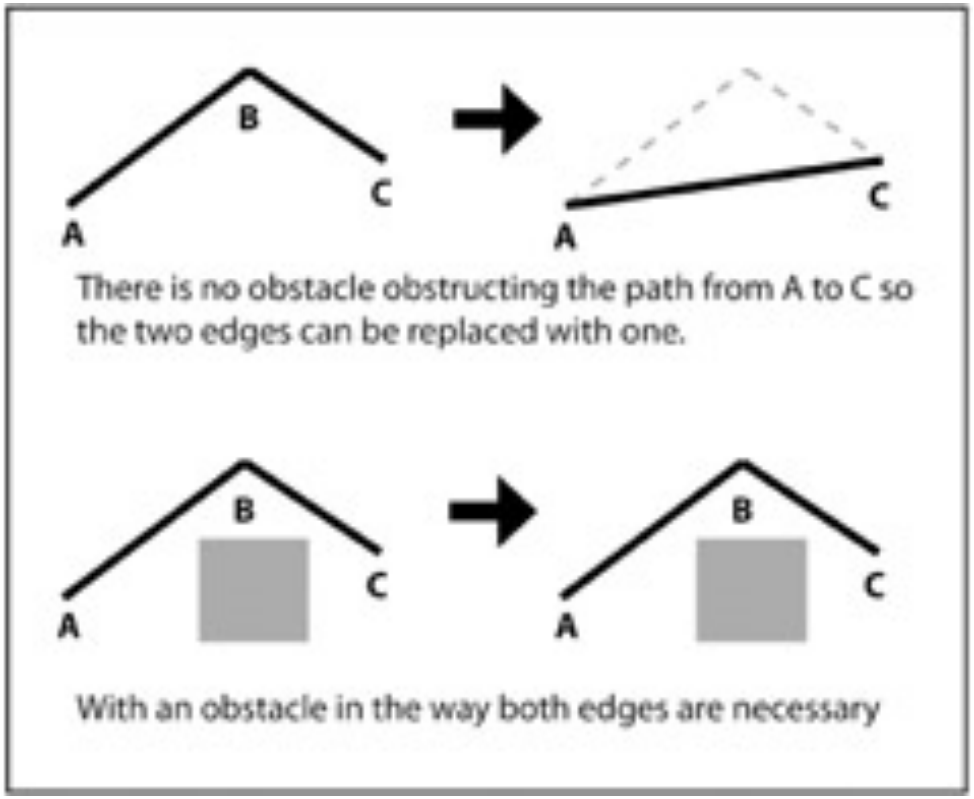
- Use A* or Dijkstra depending on whether looking for one or multiple targets

Problem: Kinky Paths



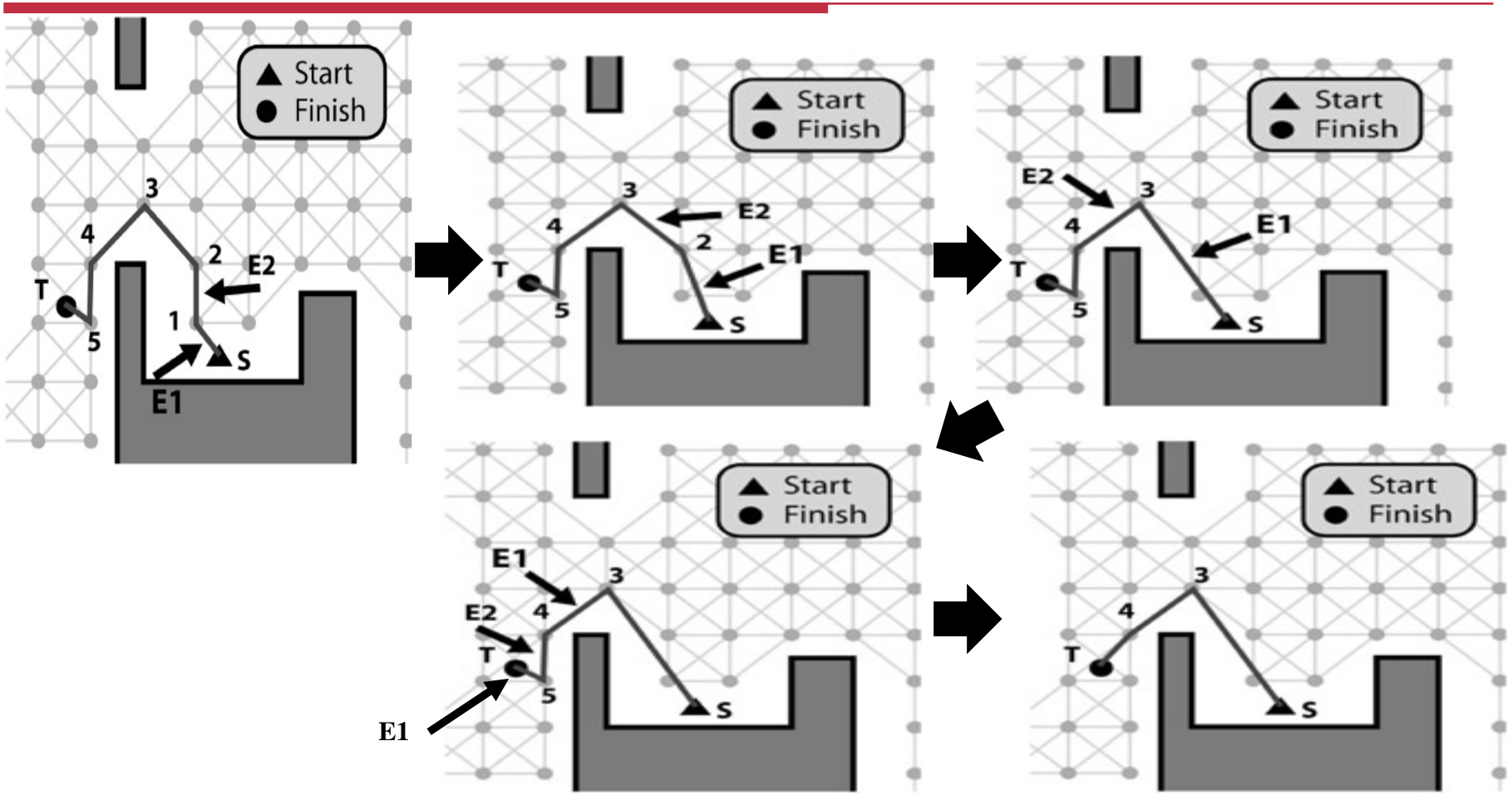
The solution: Path smoothing

Simple Smoothing Algorithm



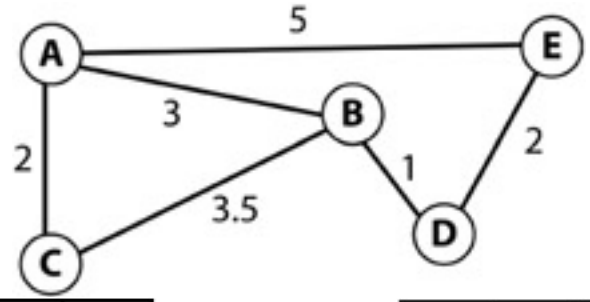
- Check for "passability" between *adjacent* edges

Smoothing Example



Methods to Reduce CPU Overhead

time/space tradeoff



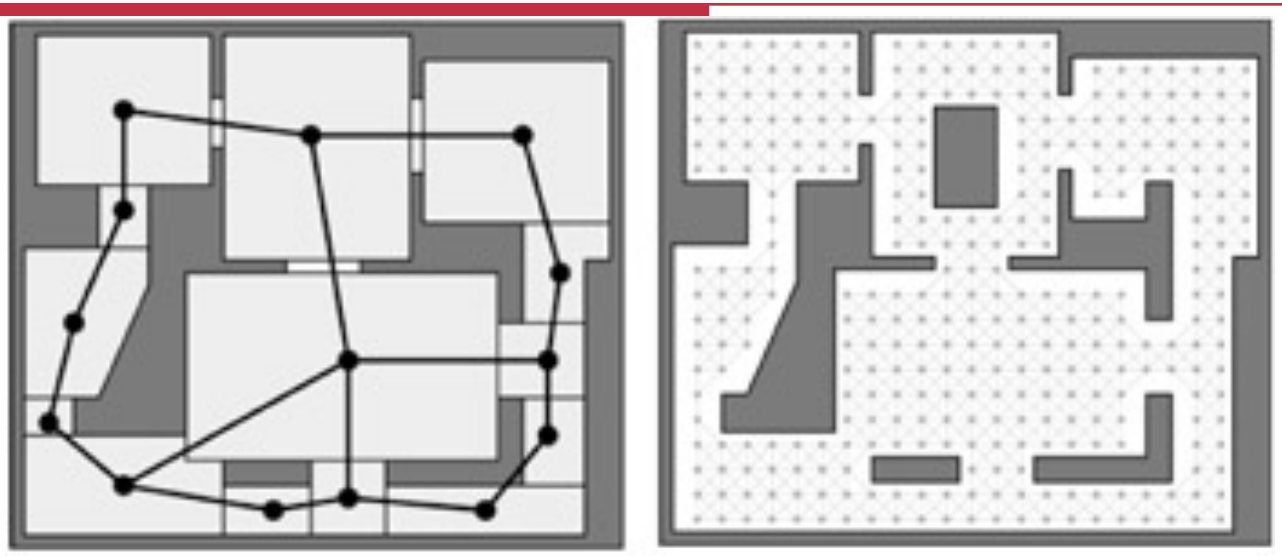
	A	B	C	D	E
A	A	B	C	B	E
B	A	B	C	D	D
C	A	B	C	B	B
D	B	B	B	D	E
E	A	D	D	D	E

shortest path table (next node)

	A	B	C	D	E
A	0	3	2	4	5
B	3	0	3.5	1	3
C	2	3.5	0	4.5	6.5
D	4	1	4.5	0	2
E	5	3	6.5	2	0

path cost table

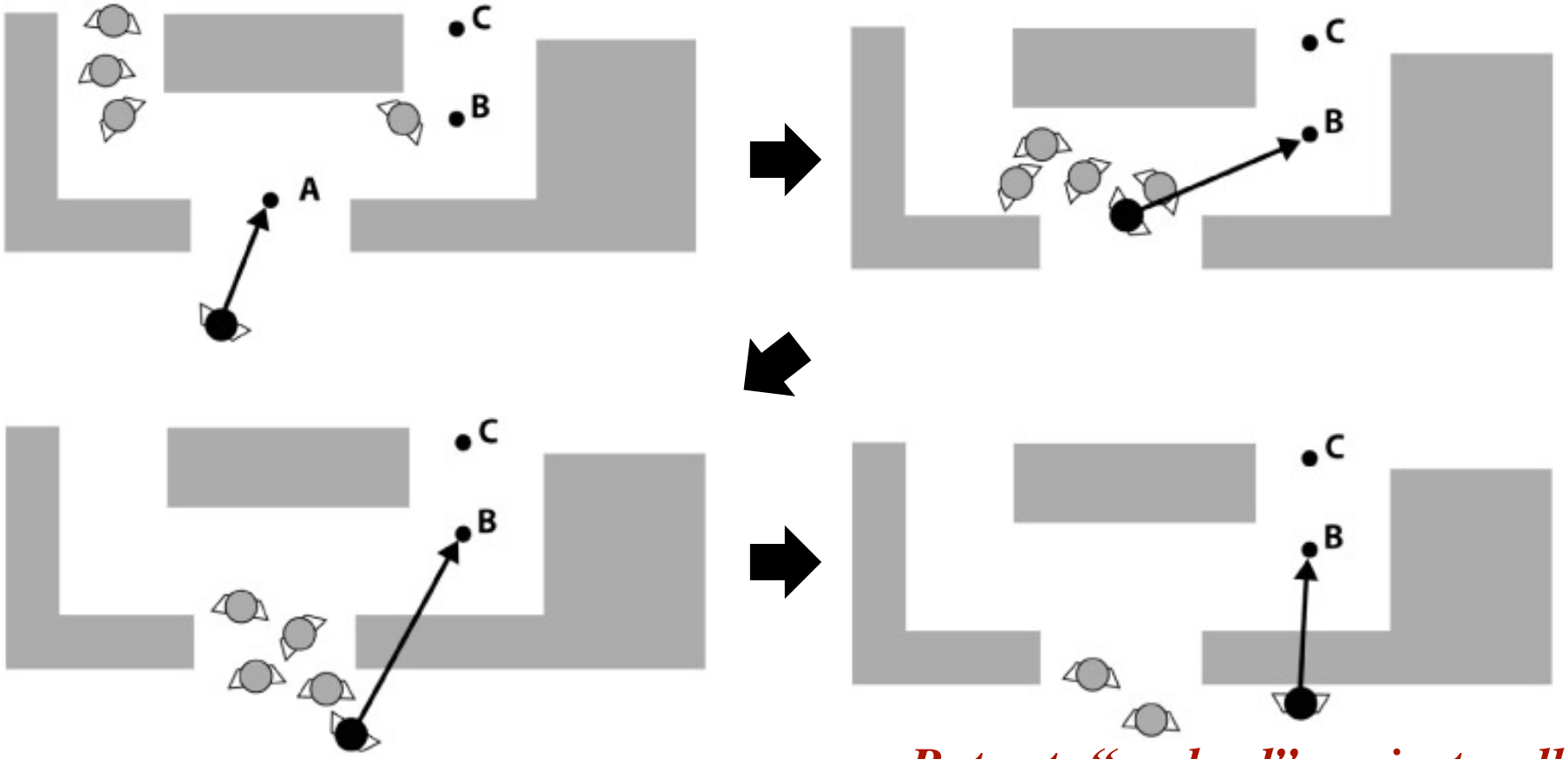
Hierarchical Path Planning



High-Level Graph Low-Level Graph

- Reduces CPU overhead
- Typically two levels, but can be more
- First plan in high-level, then refine in low-level

Getting Out of Stuck Situations



- *Bot gets "wedged" against wall*
- *Looks really bad!*

Getting Out of Stuck Situations

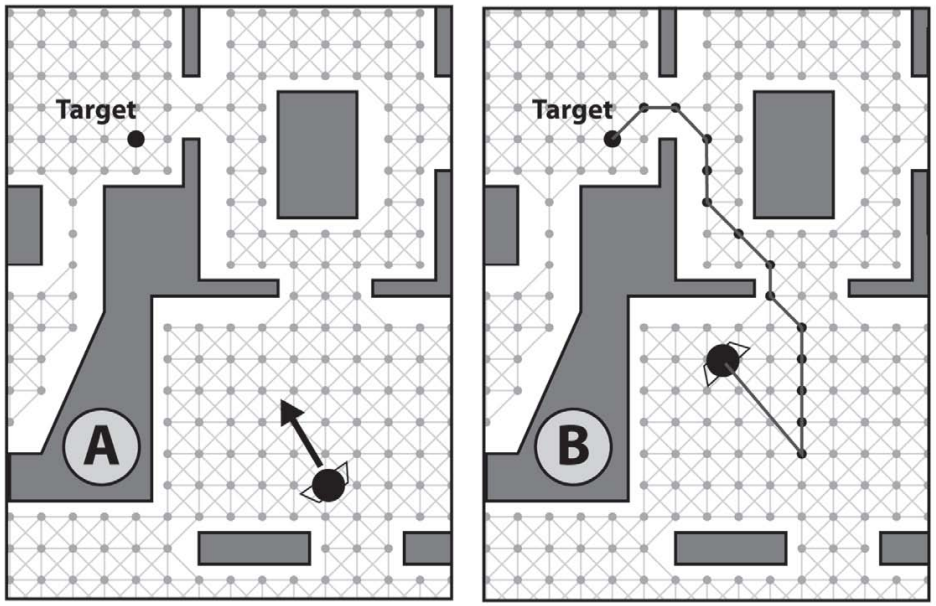
- Heuristic:
  - Calculate the distance to bot's current waypoint each update step
  - If this value remains about the same or consistently increases
    - then it's probably wedged, so backup and replan

Time Slicing -- Sketch

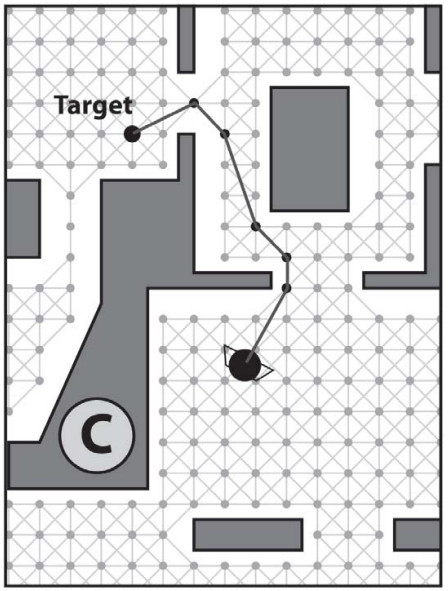
- When there are many NPC's making calls on the pathfinding module at the same time, the CPU can get dragged down...
- Solution?
  - Evenly divide fixed CPU pathfinding budget between all current callers
  - Implies that caller may have to wait for answer
- What should NPC do while it is waiting for path?
  - Do not just "block"
  - Start moving in "general direction" of target

Time Slicing and Smoothing

Smoothing is *really* needed if doing time slicing:



without smoothing



smoothed

Advanced Pathfinding Summary

- You would not necessarily use *all* of these techniques in *one* game
- Only use whatever your game demands and no more
- For reference C++ code see http://samples.jbpub.com/9781556220784/Buckland_SourceCode.zip

WPI

Thanks Chuck!

- Thanks to Chuck Rich for this material!