Path Planning Continued

CMPS 146, Fall 2013

Josh McCoy
Reading

- **Path Planning: 197-255**
- This lecture is over 237-255.
A* pathfinding review

- In A* pathfinding, nodes are locations, links are connections between locations.

- Actual costs are given by terrain (e.g. swamp harder to move through than plains).

- Heuristic (projected) costs are typically Manhattan distance (grid) or straight-line distance (non-grid movement).
Search space

- For any search algorithm, the characteristics of the search space determine performance.

- Given a search problem (start and goal), the smallest search space that contains both the start and the goal gives the best performance.
  - Though solutions generated in different search spaces will have different properties.
  - For pathfinding, the aesthetics of the path will be greatly influence by search space.

- Let’s look at different search spaces for pathfinding.
Regular Grids

http://www.codeofhonor.com/blog/the-starcraft-path-finding-hack
Regular Grids

- Won’t work for 3D game worlds without some modification
- Mostly used in strategy games (typically with a top-down perspective)
- Disadvantage: High resolution grids have large memory footprint
- Advantage: Provide random access look-up
FIGURE 2.1.3  Grid representations based on square and hexagonal cells.
Moving in only 4 cardinal directions gives you unattractive angular paths
Add in the diagonals and you improve the movement somewhat
An Optimization

- String-pulling or Line-of-sight testing can be used to improve this further
- Delete any point \( P_n \) from path when it is possible to get from \( P_{n-1} \) to \( P_{n+1} \) directly
- Don’t need to travel through node centers
Another Optimization

- Use Catmull-Rom splines to create a smooth curved path
Graphs

- The rest of the search space representations are graphs
- You can think of grids as graphs
  - Could be useful to have directed graphs (cliffs)

![Diagram of a grid and connected nodes](image)
Corner graphs

- Place waypoints on the corners of obstacles
- Place edges between nodes where a character could walk in a straight line between them
Corner Graphs

- Sub-optimal paths
- AI agents appear to be “on rails”

FIGURE 2.1.8  The path from X to Y according to a corner graph (8a) and the same path after string-pulling (8b).
Corner Graphs and String-pulling

- Can get close to the optimal path in some cases with String-pulling
- Requires expensive line-of-sight-testing
Waypoint Graphs

- Work well with 3D games and tight spaces
- Similar to Corner Graphs
  - Except nodes are further from walls and obstacles
  - Avoids wall-hugging issues

*FIGURE 2.1.10  A waypoint graph.*
Circle-based Waypoint Graphs

- Same as waypoint graphs
  - Except add a radius around each node indicating open space

- Adds a little more information to each node

- Edges exist only between nodes whose circles overlap

- Some games use a hybrid of Circle-based waypoint graphs and regular waypoint graphs, using Circle-based for outdoor open terrain and regular for indoor environments
Circular Waypoint Graphs

- Good in open terrain
- Not good in angular game worlds

FIGURE 2.1.12  A circle-based waypoint graph.
Space-Filling Volumes

- Similar to Circle based approach, but use rectangles instead of circles

- Work better than circle based in angular environments, but might not be able to completely fill all game worlds
Space-filled Volumes

Figure 2.1.13  Space-filling volumes.

Can't Get here
Navigation Meshes

- Handles indoor and outdoor environments equally well
- Cover walkable surfaces with convex polygons
- Requires storage of large number of polygons, especially in large worlds or geometrically complex areas
- Example games: used in Thief 3 and Deus Ex 2
NavMesh

- Polygons must be convex to guarantee that an agent can walk from any point within a polygon to any other point within that same polygon.

- Is possible to generate NavMesh with automated tool.
2 types of NavMeshes

- Triangle based
  - All polygons must be triangles
  - When done correctly, will not hug walls too tightly

- N-Sided-Poly-based
  - Can have any number of sides, but must remain convex
  - Can usually represent a search space more simply than triangle based (smaller memory footprint)
  - Can lead to paths that hug walls too tightly
2 types of NavMeshes

**FIGURE 2.1.14** Triangle-based (a) and N-sided-poly-based (b) navigation meshes.
N-Sided-Poly-Based

Can address this problem with post-processing
Interacting with local path following

- Search Space is static, so it can’t really deal with dynamic objects

- Should design it to give some information to pathfinding algorithm that will help

- “Can I go this way instead?” – search space should be able to answer this
Interacting with local path following

- Pathfinding algorithm must be able to deal with dynamic objects (things player can move)

- Can use simple object avoidance systems, but can break down in worlds with lots of dynamic objects
Interacting with local path following

Don't want to do this

**FIGURE 2.1.18** A dynamic obstacle (the outlined circle) along an edge in a waypoint graph.
Interacting with local path following

**FIGURE 2.1.18** A dynamic obstacle (the outlined circle) along an edge in a waypoint graph.

Should do this
Generating timely paths

- Even with a good search space, when many AI agents are pathfinding, pathfinding all at once won’t work
  - Pathfinding will chew up all the CPU resulting in game freezes
  - Even if the whole game doesn’t freeze, agents will just sit there waiting for other agents to finish pathfinding

- The solution: timeslicing
The ideal path

The idea path that would be generated by all-at-once pathfinding
The quick path is generated by limited depth A*
The full path is generated from the end of the quick path to the destination
The splice path connects from the point on the quick path when the full path became available to the full path.
The final path consists of an initial segment of the quick path, the splice path, and a final segment of the full path.
Managing pathfinding priority

- Some ways of allocating CPU resources among the pathfinders
  - Equal: each active pathfinder gets equal time (equal number of revolutions per tick)
  - Progressive: start each pathfinder with a low number of revolutions per tick – increase this (with cap) the longer the pathfinder stays active (long jobs get more resources over time)
  - Biased – give some pathfinders, like those working on quickpaths, higher priority

- If you run out of quickpath before you’ve generated the full path, you’d better crank up the priority of the full path pathfinder (otherwise your agent looks stupid just sitting there at the end of the quick path)