COMPSCI 777 S2 C 2004
Computer Games Technology

—A* Speed Optimizations—

Hans W. Guesgen
Computer Science Department

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

Motivation

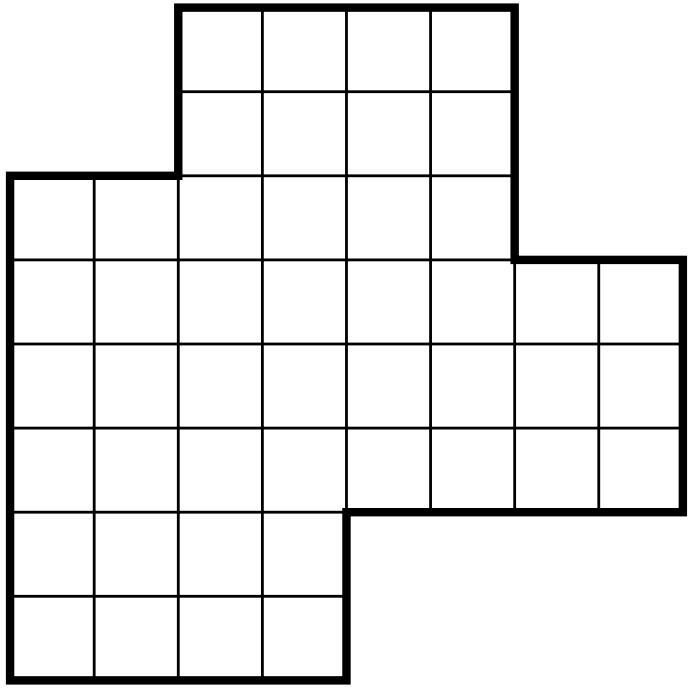
- A* is a slow algorithm that never runs as fast as you would like.
- There is a long list of optimizations you can make:
  - A* is at the mercy of the search space, so simplifying the search space is essential.
  - A* also uses a lot of memory, so optimizing memory allocation and data access is important.
  - A* demands a lot of sorting, so specialized data structures are beneficial.

Simplifying the Search Space

- The biggest win always comes from searching through less data.
- Representing the world as a simplified connectivity graph will make the A* algorithm run faster.

Rectangular or Hexagonal Grids

- A uniform rectangular or hexagonal grid overlaid onto the world.
- The size of each grid space is proportional to the size fo the smallest character.
- Therefore, a character in a grid space blocks that space during the A* search.



Pros and Cons of Rectangular or Hexagonal Grids

Pros:

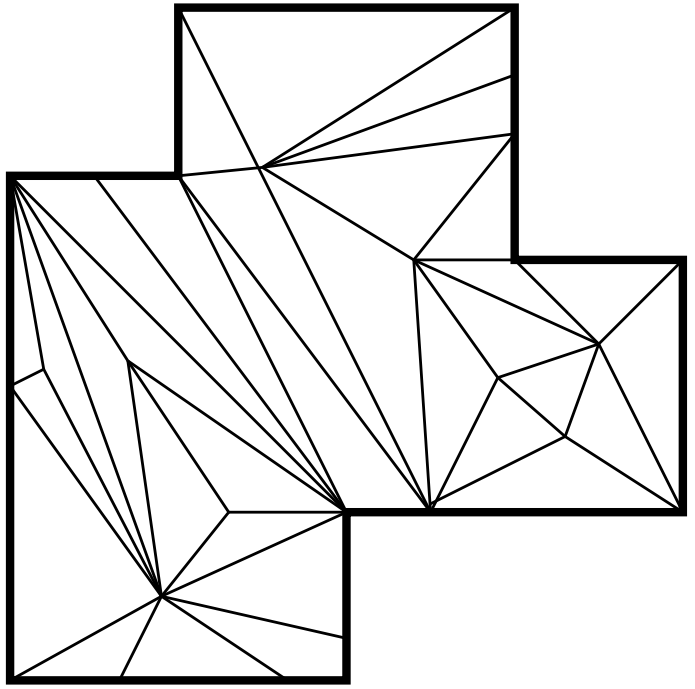
- Obstacles and characters can be easily marked in the grid allowing for avoidance.
- Works well for 2D tile-based worlds.

Cons:

- Typically results in the largest search space.
- Rectangular grids do not map very well onto 3D worlds.
- Paths tend to look like moves on a chessboard.

Actual Polygonal Floors

- In a 3D game world, the floor polygons are specifically marked and used directly as the search space.
- This polygonal floor is identical to the rendered geometry, thus being arbitrarily simple or complex.



Pros and Cons of Actual Polygonal Floors

Pros:

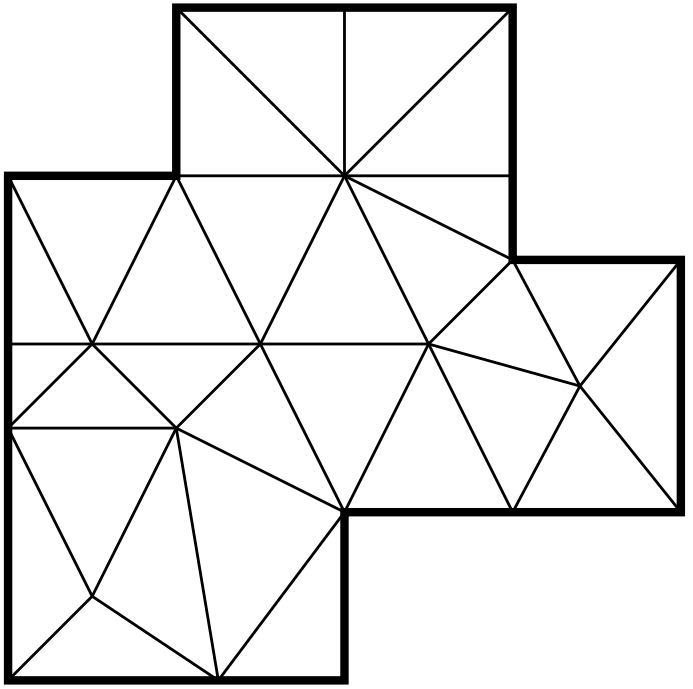
- Data structure already exists in the 3D world.
- Can be walked through quickly with a BSP tree.

Cons:

- 3D worlds can have arbitrarily high numbers of polygons on the floor.
- Cannot represent obstacles such as tables or chairs, because the floor exists beneath these objects.
- Requires algorithmic solution for choosing path points within a polygon.

Polygonal Floor Representations

- An artist or level designer creates a polygonal floor representation that is used exclusively for path finding.
- The polygons can be eliminated in places where characters are not allowed to walk, such as under tables or chairs.



Pros and Cons of Polygonal Floor Representations

Pros:

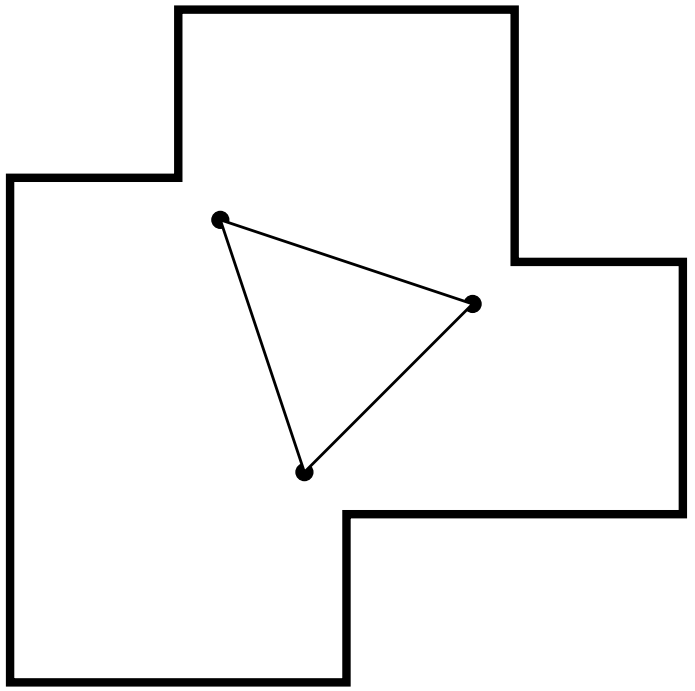
- Small search space representation.
- Can be walked through quickly with a BSP tree.
- Obstacles can be incorporated in the representaton.

Cons:

- Requires artist or level designer to construct.
- Cannot represent characters within the space.
- Requires algorithmic solution for choosing path points within a polygon.

Points of Visibility

- Points are placed at convex corners in the world, sticking out a little from each corner.
- Each point is then connected to all other points that it can "see".
- This creates a connectivity graph that describes the minimal paths required to get around walls.



Pros and Cons of Points of Visibility

Pros:

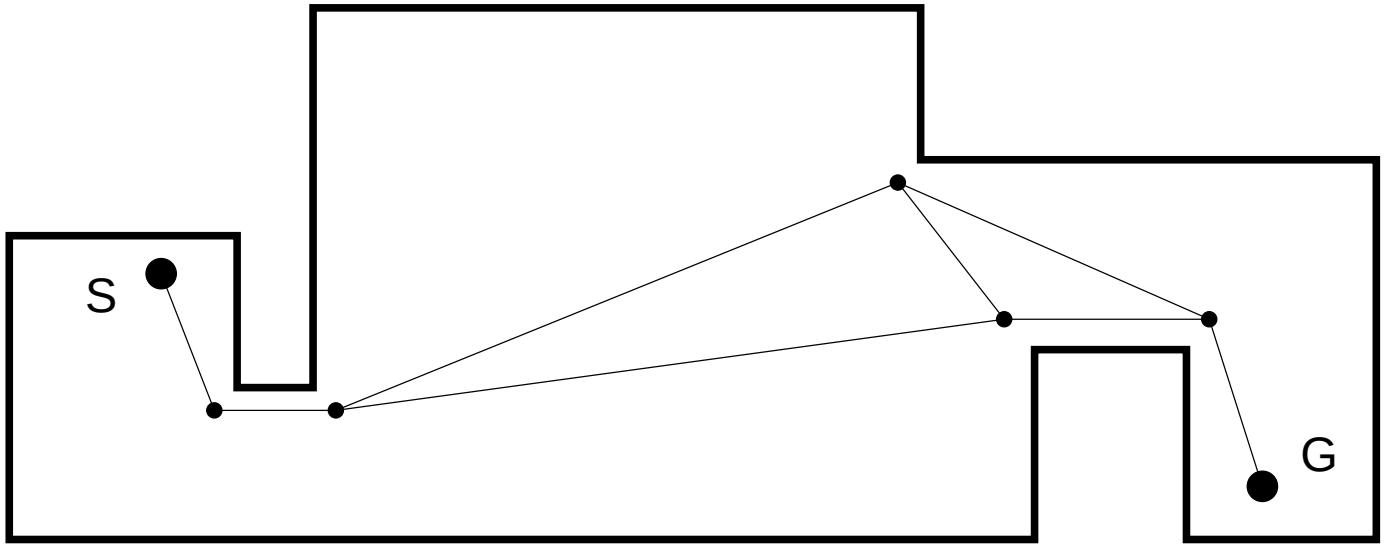
- Creates minimal search space representations.
- Obstacles can be incorporated in the representation.
- Resulting paths are perfectly direct.

Pros and Cons of Points of Visibility (cont'd)

Cons:

- Requires algorithmic or designer assistance to create the graph.
- Obstacles cannot be removed from the graph if they should be destroyed.
- Cannot represent characters within the space.
- Does not work well with entities that have large widths, such as a wide formation of characters.
- Worlds with curved walls could cause the graph to become unnecessarily complex.

Example of A* with Points of Visibility



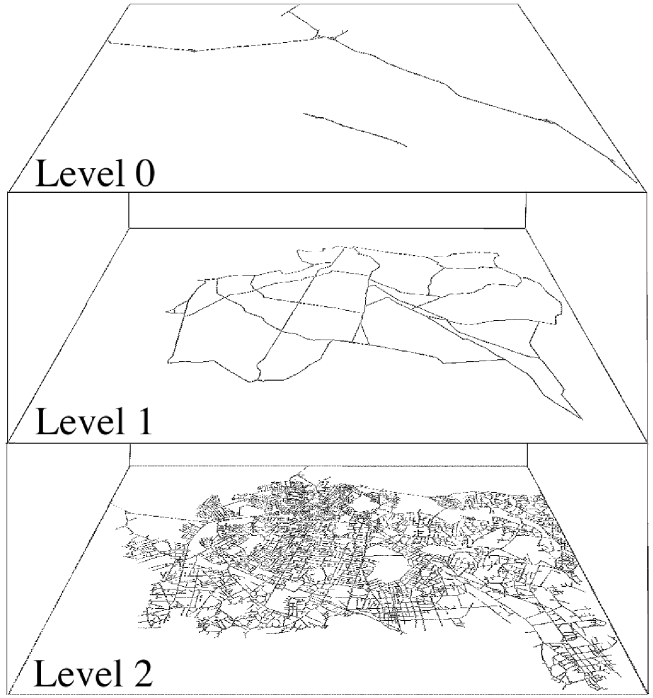
Hierarchical Path Finding

- Hierarchical search reduces the complexity of search by using a series of searches among levels of smaller size in a hierarchical problem space.

- The idea of hierarchical search in the context of path finding:

  1. Search in the base space to the nearest node in the abstract space.
  2. Search in the abstract space from the terminating node of the previous search to the nearest node in the abstract space to the goal node.
  3. Search from the terminating node in the previous search to the goal node in the base state space.

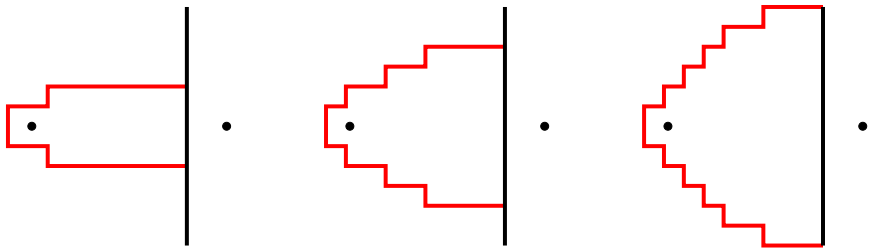
Hierarchical Road Map of Auckland



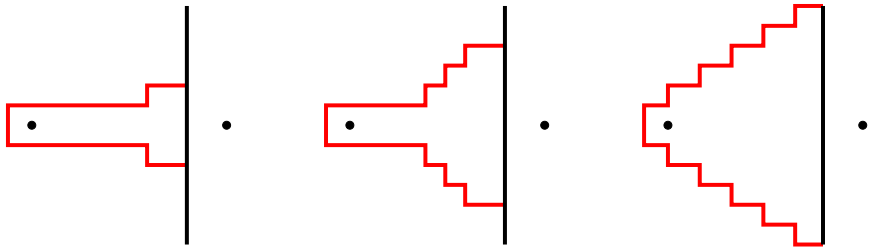
Overestimating the Heuristic Cost

- Using a heuristic that routinely overestimates by a little usually results in faster searches with reasonable paths.
- If the heuristic part of the total cost is bigger than it should be, it distorts the reasoning by which nodes on the Open list are picked off.
- Since A* always picks the node with the least total cost, this distortion promotes nodes closer to the goal to be picked.

Example of Overestimating the Heuristic Cost



Non-overestimating heuristic



Overestimating heuristic